NAVAL POSTGRADUATE SCHOOL
Monterey, California

19990419 039

THESIS

IMPLEMENTATION OF A PORTABLE PSDL EDITOR
FOR THE HETEROGENEOUS
SYSTEMS INTEGRATOR

by

Ilker Duranlioglu

March 1999

Thesis Advisor: Man-Tak Shing

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1999	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE IMPLEMENTATION OF A PORTABLE PSDL EDITOR FOR THE HETEROGENEOUS SYSTEMS INTEGRATOR	5. FUNDING NUMBERS
6. AUTHOR(S) Duranlioglu, Ilker	

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)	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

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 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 Computer Aided Prototyping System (CAPS) is an integrated set of tools that is used for rapid prototyping of real time systems. Prototype System Description Language (PSDL) is the prototyping language that captures the requirements and produces executable prototypes. Prototypes can be created by using specially designed PSDL Editor, which can automatically generate PSDL code from data flow graphs. The Heterogeneous Systems Integrator (HSI) is an extension to CAPS, designed to automate the process of integrating complex distributed systems, where the subsystems can reside on different locations, be implemented in different hardware, operating systems and programming languages.

It is envisioned that the HSI will be a distributed system itself. Users at remote sites need not install the entire HSI system, but only the User Interface for entering the PSDL specification of the target systems. This research is the first step in the evolution of HSI. The focus is to create a portable user interface, which can be used in any environment (hardware and operating system).

We have designed and implemented a platform independent HSI user interface using the Java programming language. The functionalities of CAPS Release 2.0 PSDL Editor are mainly preserved in this implementation with a few added features. The new editor shows significant improvement in performance and user friendliness over the previous versions of CAPS PSDL Editor.

14. SUBJECT TERMS Rapid Prototyping, User Interface	15. NUMBER OF PAGES 304
	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

IMPLEMENTATION OF A PORTABLE PSDL EDITOR
FOR THE HETEROGENEOUS
SYSTEMS INTEGRATOR

Ilker Duranlioglu
Lt.J.G. Turkish Navy
B.S., Turkish Naval Academy, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1999

Author: _____
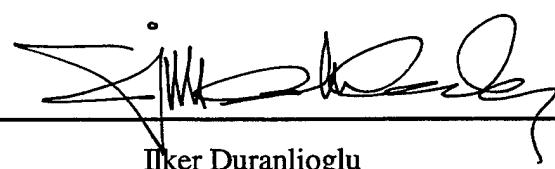
Ilker Duranlioglu

Approved by: _____
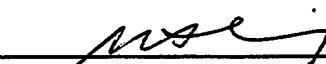
Man-Tak Shing, Thesis Advisor

_____
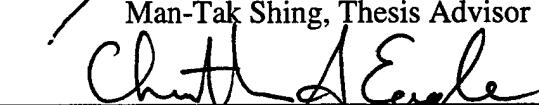
Chris Eagle, Second Reader

_____
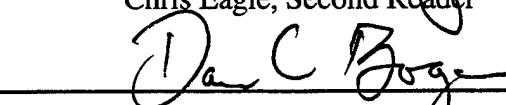
Dan Boger, Chairman
Department of Computer Science

iii

ABSTRACT

The Computer Aided Prototyping System (CAPS) is an integrated set of tools that is used for rapid prototyping of real time systems. Prototype System Description Language (PSDL) is the prototyping language that captures the requirements and produces executable prototypes. Prototypes can be created by using specially designed PSDL Editor, which can automatically generate PSDL code from data flow graphs. The Heterogeneous Systems Integrator (HSI) is an extension to CAPS, designed to automate the process of integrating complex distributed systems, where the subsystems can reside on different locations, be implemented in different hardware, operating systems and programming languages.

It is envisioned that the HSI will be a distributed system itself. Users at remote sites need not install the entire HSI system, but only the User Interface for entering the PSDL specification of the target systems. This research is the first step in the evolution of HSI. The focus is to create a portable user interface, which can be used in any environment (hardware and operating system).

We have designed and implemented a platform independent HSI user interface using the Java programming language. The functionalities of CAPS Release 2.0 PSDL Editor are mainly preserved in this implementation with a few added features. The new editor shows significant improvement in performance and user friendliness over the previous versions of CAPS PSDL Editor.

TABLE OF CONTENTS

I. INTRODUCTION

The classic Waterfall Model for software development consists of five phases (Figure 1). Among these, the Requirements Analysis phase has special importance. The development efforts start with the description of requirements. This is generally not very easy, because few customers use formal descriptions and frequently, they may not be able to fully describe what they want. It is the job of software developers to discover missing or forgotten requirements, and to understand what the customer actually wants to see in the end. The product may be perfectly error free, but it is useless if it does not reflect the needs of the customers.

Rapid software prototyping is an iterative software development methodology and its purpose is to improve the analysis, design and the development of software systems [Ref. 1].
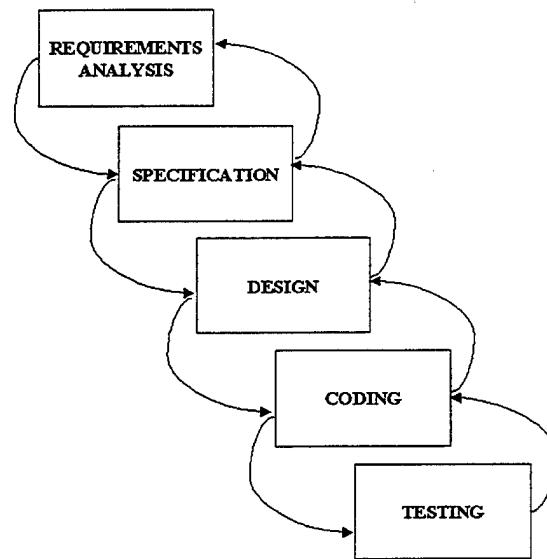
Figure 1. Waterfall Model

A prototype is a simplified model of a proposed system [Ref. 2]. Prototypes improve customer–developer communication by demonstrating the feasibility, behavior or performance of the desired product. Hence, the customer can have a better view of his/her needs. This helps developers in identifying the requirements via the feedback from the customers.

A. COMPUTER AIDED PROTOTYPING SYSTEM (CAPS)

Computer Aided Prototyping System (CAPS) is a prototyping tool that is developed by the Naval Postgraduate School Software Engineering Group. CAPS provides automated generation of executable prototypes. The prototype, in turn, helps in capturing the requirements of the end user.

Real time systems are generally very complex and hard to analyze. CAPS provides tools that has been specifically designed to support prototyping of such complex systems. CAPS has easy to use visual tools that capture the high level properties of systems, which in turn produce a formal description in a high-level prototyping language and then generate executable code.

The tools that CAPS uses are Editors, Software Base, Execution Support and Project Control (Figure 2). The prototyping language that CAPS is based upon is the Prototype System Description Language (PSDL).

PSDL is a high level language which helps to specify prototypes. The system is decomposed into subsystems using data flow diagrams. Data streams provide the communication between the subsystems.

The editors that CAPS provides are the PSDL Editor, Source Code Editor and GUI Interface Editor. The primary editor is the PSDL Editor, which is specifically designed to build PSDL prototypes. The prototypes are viewed as augmented data flow graphs in the PSDL editor. As the prototype is being built using the PSDL Editor, the data flow graph is translated automatically into PSDL code in the background and syntactic checking of the prototype is performed as well. The Source Code Editor is used to develop the software packages that will be used in the prototype and the GUI Interface

2

```
                        ┌─────────────────┐
                        │      CAPS       │
                        └─────────────────┘
        ┌──────────────┬──────┴──────┬──────────────┐
  ┌──────────┐  ┌──────────┐  ┌──────────┐  ┌──────────┐
  │ EDITORS  │  │ SOFTWARE │  │EXECUTION │  │ PROJECT  │
  │          │  │   BASE   │  │ SUPPORT  │  │ CONTROL  │
  └──────────┘  └──────────┘  └──────────┘  └──────────┘
```

PSDL Editor   Translator   Merger

Ada Editor   Scheduler   Evolution Control

TAE Interface   Compiler
Editor
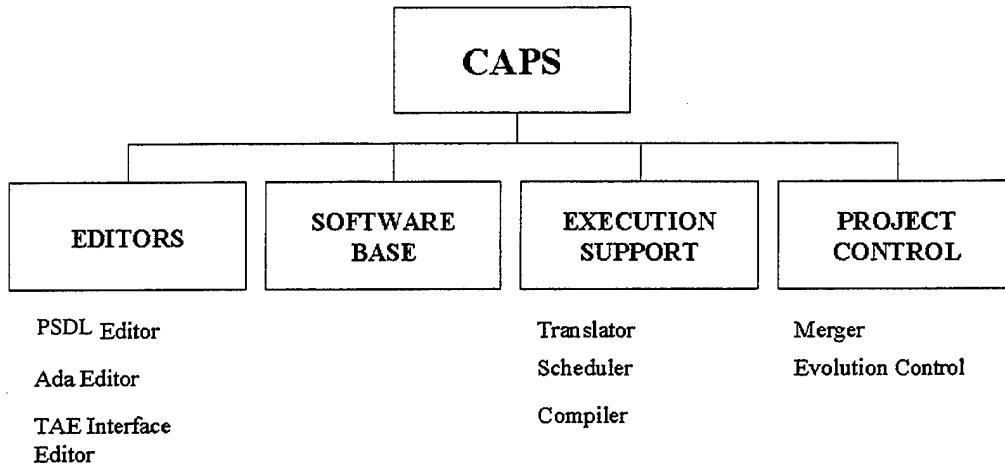
Figure 2. CAPS Subsystems

Editor helps to build Graphical User Interfaces for the prototype.

The Software Base contains reusable prototype components. These components can be retrieved by queries or by browsing the repository once the database is established. This feature saves time by reusing existing components rather than rewriting them each and every time. In the current version of CAPS, the Software Base is not yet implemented.

The Execution Support system contains automated tools to rapidly construct prototypes of real time systems. It consists of a translator, a scheduler, and a compiler. The translator generates code that binds the reusable components that are taken from the software base. The scheduler is invoked before the execution of the prototype to arrange appropriate time slots for the components that have real time constraints. If the scheduler

successfully completes its execution, all of the time-critical operators are guaranteed to meet their timing requirements even in worst case scenarios [Ref. 3]. The scheduler also produces a low-priority dynamic schedule that runs during the execution of the prototype and allocates time slots for components that do not have real time constraints. These components are scheduled to run during the time slots that are not scheduled for any of the time critical operators. Finally, the compiler compiles the generated source code into executable binary code.

B.   CAPS RELEASE 2.0 GRAPH EDITOR

The current PSDL Editor of CAPS is mostly implemented by Kenneth Moeller in his thesis research [Ref. 4]. The implementation consists of two parts. The first part is the user interface used to build the prototypes. The users of CAPS interact only with this interface. A prototype is built by using data flow diagrams and the graph editor automatically translates the diagrams into PSDL code. The second part of the editor is a background checker, which is actually a parser to check the syntactic correctness of the PSDL code. The parser is invoked following certain events such as changing the level in the hierarchy of the prototype or saving the prototype into disk.

Figure 3 shows the PSDL Editor of CAPS. It basically consists of a Menu Bar, a Tool Bar and a drawing area for the PSDL prototypes. This section will very briefly describe the graph editor and its functionalities.

The drawing area can be thought of as a very limited drawing application. It is used to build the PSDL prototypes. The prototypes are built as a hierarchy of operators. An operator represents the subsystems of the intended software. Only one level of the hierarchy can be viewed and edited at a time. The user can perform certain operations in the drawing area such as inserting an operator, inserting a data stream, editing the properties of operators and streams and moving the location of the operators and streams.

The Tool Bar has eight buttons that choose the operation to be performed in the drawing area. Selecting the appropriate button from the toolbar and clicking into the drawing area will draw an operator or a data stream. The select button allows selection
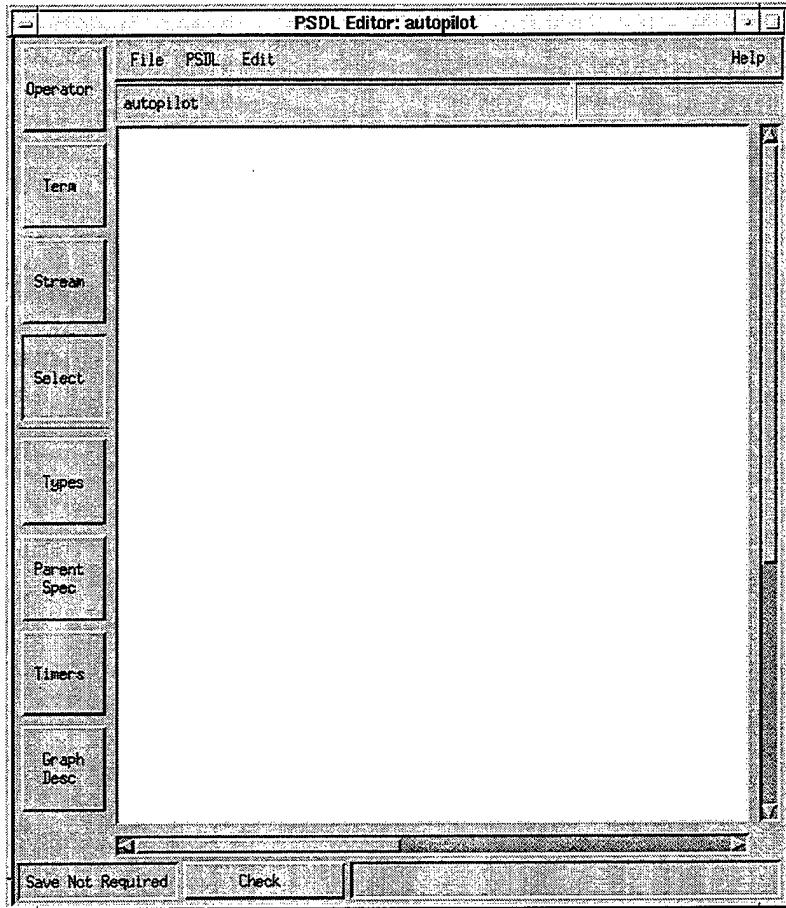
Figure 3. CAPS Graph Editor

of an operator or a data stream within the drawing area.

The Menu Bar consists of four menus. The file menu involves actions to save the prototype to the disk, restore a prototype from the disk, abandon changes that have been made, print the diagram and exit the PSDL editor. The PSDL menu has actions to change the level in the hierarchy by either decomposing the current operator or shifting to the parent or the root operator. Users can change colors or fonts from the edit menu. Undeleting an operator is also available from the edit menu. Finally, the help menu includes online information about PSDL concepts that can be browsed very quickly.

It is possible to change the properties of an operator. A pop-up dialog (Figure 4) is opened when the user clicks on an operator with the right mouse button. It allows user to
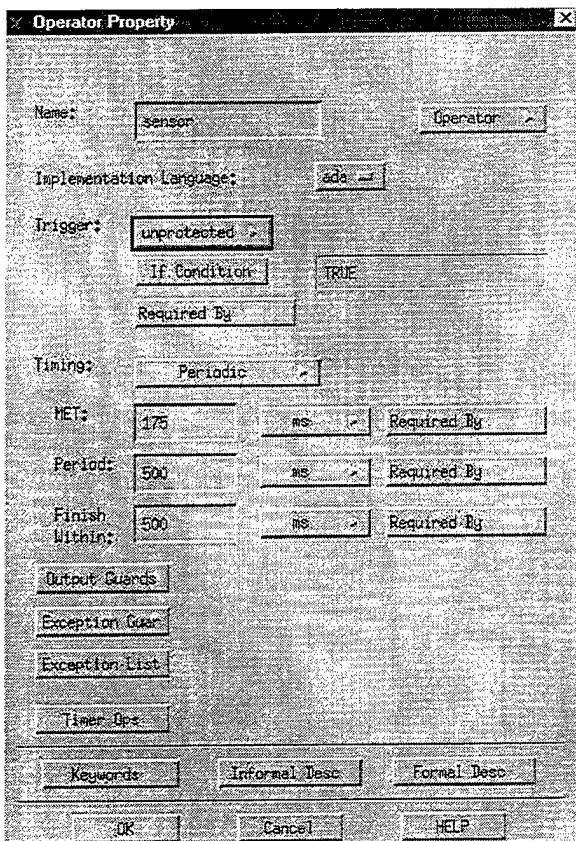
Figure 4. Operator properties pop-up menu
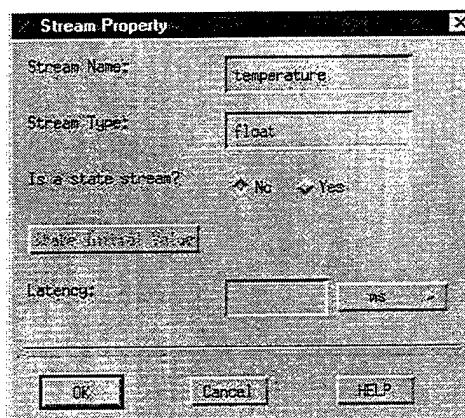
Figure 5. Stream properties pop-up menu

6

change various properties and timing requirements for the operator. A similar but simpler pop-up dialog is available for the properties of data streams as well (Figure 5).

C. RESEARCH GOAL

The rapid prototyping of CAPS will be extended in future into a Heterogeneous Systems Integrator (HSI), which will be used to automate the process of integrating complex distributed systems. The requirements for HSI are not specified as of the time of this research. The main idea is to have HSI reside on a server. Clients will enter their PSDL specification of the system using a PSDL Editor, send the PSDL specification to the server and receive back an executable prototype. Users need not install the entire HSI system, only the PSDL Editor will be needed to build prototypes at the remote site.

The main disadvantage of the current CAPS PSDL Editor is that it is implemented in C++ and can only be executed under Unix environments. A portable implementation of the PSDL editor is needed to build PSDL prototypes under any environment. The goal of this research effort is to overcome these limitations by reimplementing the PSDL Editor using a platform independent programming language.

We have chosen the the Java* programming language for this purpose. Java is a programming language developed by Sun Microsystems and has gained popularity in recent years with the developments in the Internet. Java allows programs to run on different platforms without recompiling them. The main reason why Java is so popular is that small Java programs called Applets can be used to enhance web pages over the internet.

The current CAPS PSDL editor is the basis for our implementation. However, it is not a one-to-one translation from one language to another one. Some new features are added as well. These will be introduced in Chapter III and Chapter IV in detail.

---

* Java is a trade mark of Sun Microsystems.

II. REDESIGNING CAPS MAIN PROGRAM

The Heterogeneous Systems Integrator (HSI) PSDL Editor is not a stand-alone program. To edit an existing prototype or to create a new prototype, the HSI main program must be executed first. The PSDL Editor is invoked from the main program. Thus, together with the PSDL Editor, the main program also had to be reimplemented. The new implementation is very similar to CAPS Release 2.0, except that it currently only allows to open/create a prototype and to invoke the PSDL Editor on that prototype.

A. PROTOTYPE DIRECTORY STRUCTURE

Prototypes are placed in a directory structure similar to the CAPS Release 2.0. The program will look for the "PROTOTYPEHOME" environment variable first. If this variable is set as a command line argument, the program will use that directory to find existing prototypes or to create new prototypes. If it cannot find such a variable, then the user home directory will be used as the default "PROTOTYPEHOME".

Prototypes will be placed in the ".caps" sub directory under the "PROTOTYPEHOME" directory. A new sub directory will be created under ".caps" directory for each prototype having the same name as the prototype. Each prototype directory will have different versions of the same prototype as "1.1", "1.2" and so on.

B. CREATING A NEW PROTOTYPE

Selecting the "New" menu item under the "Prototype" menu will launch an input dialog asking the user to input the name and the version of the prototype. This dialog provides two text boxes to enter the name of the prototype and the version of the prototype. If the user leaves the version text box blank, the version number for the prototype will be automatically accepted as 1.1.

A new directory will be created under ".caps" directory containing the prototype name that is entered by the user. The version number is also created as a directory under the prototype

9

directory. This directory will contain the PSDL prototype file, which will be created by the PSDL Editor.

The user may enter a prototype name with a version number that corresponds to an existing prototype. This may be due to a mistake or that he/she may want to overwrite an existing prototype. In this case, HSI will issue a warning message telling the user that they are about to overwrite an existing prototype. The operation may then be continued or cancelled.

C.   OPENING AN EXISTING PROTOTYPE

When the "Open" menu item is selected under the "Prototype" menu, the program will look for existing prototypes under the ".caps" directory and list them in a selection box. A different selection will exist in the selection box for different versions of the same prototype.

D.   INVOKING THE GRAPH EDITOR

After creating a new prototype or selecting an existing prototype, the user can open the PSDL Editor by selecting "PSDL" menu item from the "Edit" menu. If no prototype is selected, the program will show an error message.

It is possible to edit more than one prototype at the same time, however the program will not allow more than one instance of the same prototype at the same time. An attempt to run the PSDL Editor with the same prototype will result in an error message.

E.   EXITING THE PROGRAM

The program will exit either when the "Quit" menu item is selected from the "Prototype" menu, or when the user clicks on the window close icon. In both cases, the program will prompt the user if the current prototypes are not saved. The user can choose to save the prototype, cancel the 'quit' operation or close the program without saving the prototypes.

III. USER INTERFACE DESIGN

The main idea behind this implementation was to build a PSDL editor that was platform independent. We tried to capture all the functionalities of the PSDL Editor that was used in CAPS Release 2.0. PSDL mappings from and to the PSDL Editor are the same. Some new features are added to provide a user-friendlier interface. LCDR. Chris Eagle created a preliminary Java version of the CAPS PSDL Editor that had both application and applet versions. We adopted some of his ideas and most of his class hierarchy in this implementation. The complete source code for this implementation can be found in Appendix D. The documentation of the source code is created by using Javadoc and can be found in Appendix C.

A. ARCHITECTURE OVERVIEW

The HSI main program runs as one thread of execution. Each PSDL Editor that is launched by the HSI main window has a separate thread of execution as well. As described in Chapter 2, it is possible to edit more than one prototype at the same time. However, no two prototypes can be launched using the same prototype file. There are no shared variables used by these threads. Thus, no synchronization procedure is necessary among the threads and none is provided.

1. Program Packages

The program consists of six packages and several classes in these packages. The classes are packaged according to their functionality in the program. Figure 6 shows the hierarchy of the packages. The packages also represent the directory structure of the program.

The images that are used in the program are under the caps.Images directory. They are referred to by using this directory structure in the program.

a. *Package caps.Builder*

Package caps.Builder contains the classes that are responsible for reading a PSDL prototype file and to construct the data structures that are used to represent the data flow diagram

caps

caps:Builder
- ASCII_CharStream
- ParseException
- PSDLBuilder
- PSDLBuilderConstants
- PSDLBuilderTokenManager ExecSupportMenu
- Token
- TokenMgrError
- PsdlBuilder.jj

caps:CAPSMain
- CAPSMainMenuBar
- CAPSMainWindow
- DataBasesMenu
- EditMenu
- ExitCAPSMain
- HelpMenu
- PrototypeMenu

caps:Display
- DisplayComponent
- DisplayExternal
- DisplayVertex
- EdgePath

caps:GraphEditor
- ColorConstants
- DrawPanel
- EdgeProperties
- Editor
- EditorMenuBar
- ExitEditor
- FontConstants
- GE_EditMenu
- GE_FileMenu
- GE_PSDLMenu
- GE_ViewMenu
- IdListEditor
- Popup
- PrintJob
- StatusBar
- TextEditor
- ToolBar
- TreePanel
- TreePanelRenderer
- VertexProperties

caps:Parser
- ASCII_CharStream
- CreatePsdl
- GrammarCheck
- ParseException
- PsdlParser
- PsdlParserConstants
- PsdlParserTokenManager
- Token
- TokenMgrError
- PsdlGrammar.jj

caps:Psdl
- DataFlowComponent
- DataTypes
- Edge
- External
- PSDLTime
- Vertex

caps:Images
- Gif images

Figure 6. HSI Packages

12

in the PSDL Editor. PSDLBuilder.jj is the JavaCC source file that is used to create the classes in the package.

PSDLBuilder is the program that parses the PSDL file and creates the data structures. The other classes are used for functionalities such as providing tokens. PSDLBuilder contains the semantic actions, which are actually embedded Java code that creates the prototype.

As the classes in this package are created by JavaCC, we did not provide any documentation for these classes in Appendix C. The source code is quite difficult to read. Because of that, it is more convenient to read the PSDLBuilder.jj file instead of the others. Any change to the classes in this package should be first made to PSDLBuilder.jj. Then, it must be recompiled by JavaCC.

b. *Package caps.CAPSMainWindow*

This package contains the main HSI program that is necessary to run the PSDL Editor.

c. *Package caps.Display*

Display package contains the classes that are used to model the graphical representation of the PSDL prototype. The data structures that are used in this package are explained later.

d. *Package caps.GraphEditor*

GraphEditor package contains the classes that implement the user interface components of the PSDL editor.

e. *Package caps.Parser*

The PSDL parser is created by PSDLParser.jj. JavaCC is used to create the parser files from PSDLParser.jj. The parser is used within the PSDL editor to validate user inputs. If the user input is not accepted by the parser, a ParseException is thrown. This is reflected to the user by an error message. The parser accepts the user input if it does not violate the grammar rules.

The grammar rules that are used to create the parser are the same as Appendix A. Appendix B contains PSDLParser.jj and PSDLBuilder.jj for reference. Like the PSDLBuilder package, no documentation for the classes of this package is provided. The source code of this package is difficult to trace. We recommend to refer to PSDLParser.jj file. Again, any changes to

the source code must be made via the PSDLParser.jj file. This file then must be compiled by JavaCC to create the parser.

f.   Package caps.PSDL

The caps.PSDL package contains the classes that implement the data structures that are used to represent the PSDL prototype.

2.   The Architecture

The HSI program consists of the main program and the PSDL editor. The PSDL editor must be launched from the main program. The entire program is written in Java, and is platform independent. It can be executed on any machine or operating system where a Jdk1.2 compatible Java runtime environment can be found. As Java does not allow global variables, each class has its own fields and methods that is written in an object-oriented way.

Figure 7 shows the data flow between the major modules of the program. The main program invokes the PSDLBuilder, which reads the prototype from file and constructs the data structure. The main program provides the prototype data structures to the PSDL Editor. The data flow diagrams are constructed by the PSDL Editor routines. PSDL Parser validates user inputs during the modification of the prototype. CreatePsdl routine maps the data structures to PSDL and saves the prototype to disk.

B.   DATA STRUCTURES

The PSDL Editor maintains two kinds of data structures. One of these data structures is a tree that represents the prototype components as nodes in the tree. If an operator is composite, operators corresponding to the vertices in its data flow diagram implementation are represented as its children nodes. If the component is an atomic operator or a stream, then it is a leaf node in the tree hierarchy.

The other data structure is a vector that holds display components as its components. These display components are the current operator's children components. As the user navigates through the prototype, this vector is updated continuously to hold the children of the current operator. The data structures and their hierarchy are shown in Figure 8.

14
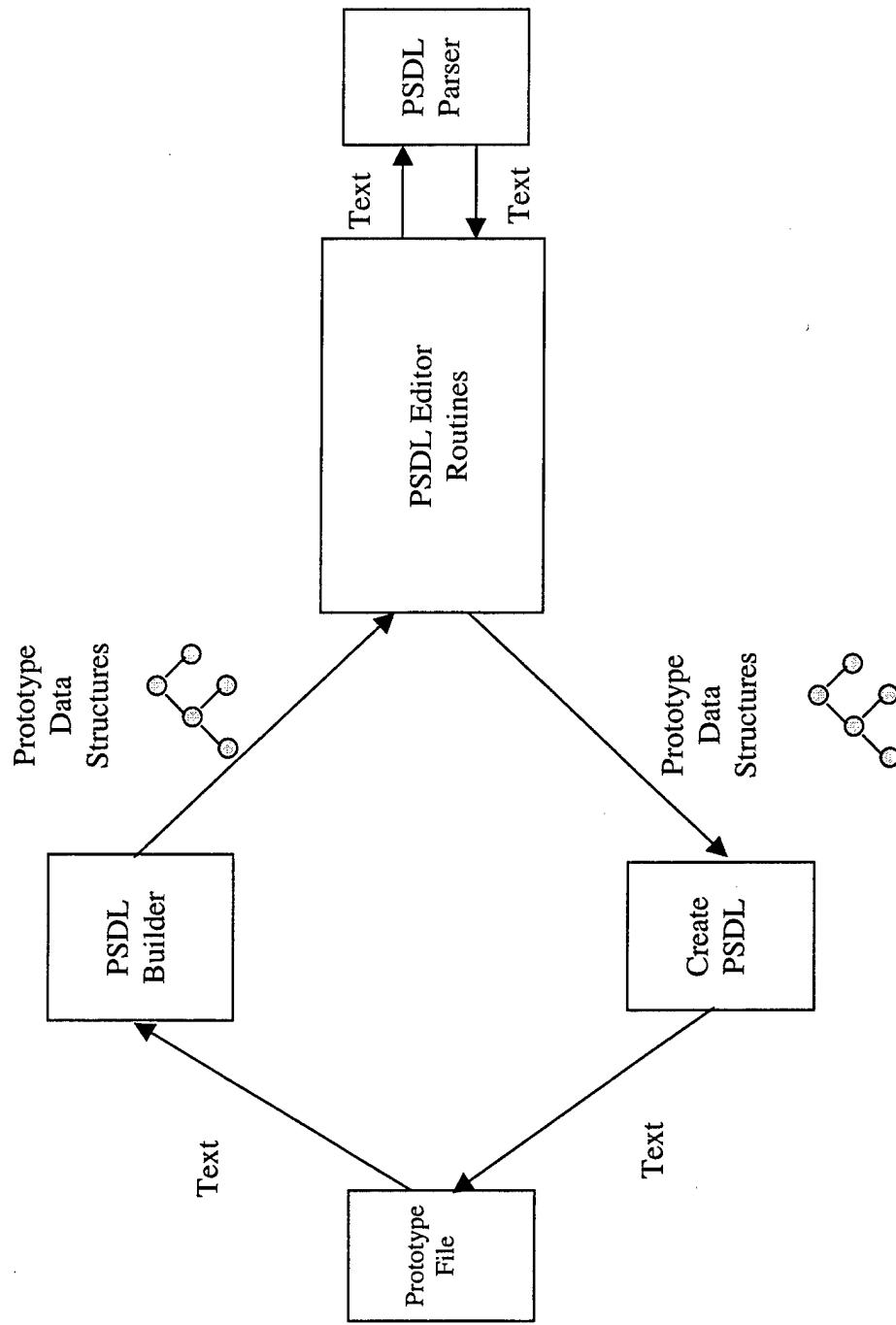
Figure 7. Data Flow Between Major Modules
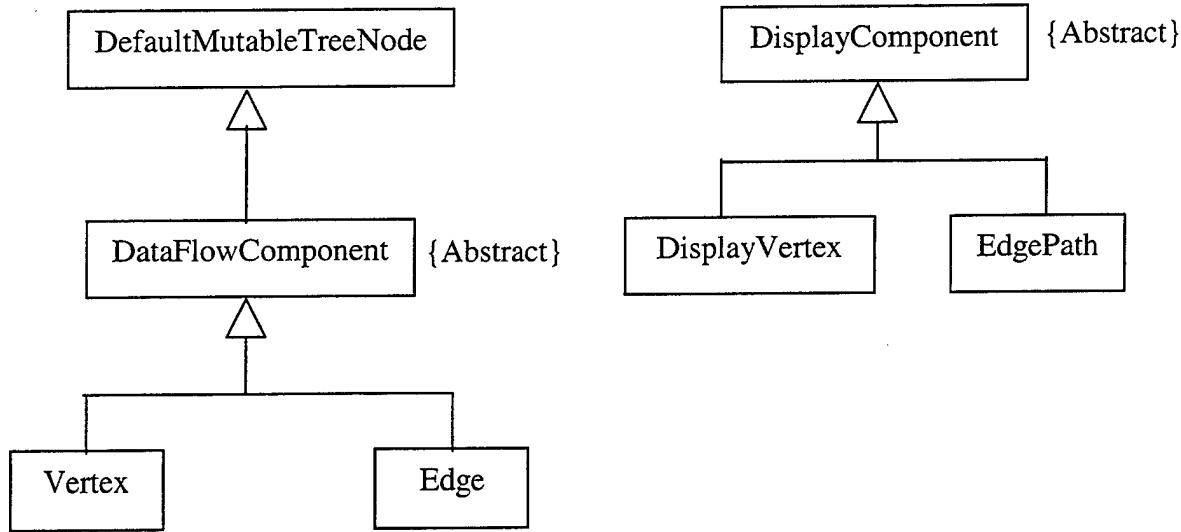
Figure 8. DataFlowComponents and DisplayComponents Hierarchy

1. Data Flow Components

A DataFlowComponent is the parent class of an Edge and a Vertex. The common fields of Vertex and Edge are abstracted in the DataFlowComponent. DataFlowComponent itself is implemenented as a subclass of DefaultMutableTreeNode.

2. Display Components

DisplayComponents objects are used to represent Data Flow Diagram Components visually on the drawing panel. DisplayComponent is an abstract class and contains the common fields and methods of the DisplayVertex and EdgePath classes.

DisplayVertex class has a Vertex field. It does not contain any information about the PSDL components. To view a Vertex on the drawing panel, it queries necessary fields from its associated Vertex field. These are values such as x and y locations of the component, color of the component, etc. The shape field of DisplayVertex holds either a rectangle object or a circle object according to the type of the Vertex.

EdgePath class is implemented similar to the DisplayVertex class. Its shape object is implemented as a GeneralPath object from the java.awt.geom package. It also queries values

from its associated Edge object to display the component on the drawing panel.

C.   PSDL MAPPING

The mapping of PSDL from and to the Editor is implemented in the same way as CAPS Release 2.0. Chapter 4 of Reference 4 describes these mappings. The focus of the Editor is again on the current operator. All modifications are performed on the current operator except the global data types and the properties windows for the child operators and streams.

PsdlBuilder package is responsible for mapping PSDL to the Graph Editor data structures. CAPS Release 2.0 maintained two copies of the data structures, one for the Graph Editor and one for the Background Checker. This redundancy crippled the performance of the program. This implementation maintains one copy of these data structures throughout the program. This improves the performance as no synchronization is necessary for the redundant copy.

CreatePsdl class maps the PSDL Editor data structures to PSDL. It starts from the root operator and visits all the children and sub-children. While visiting an operator, it extracts the fields of the operator and creates PSDL code.

D.   SYNTACTIC VALIDATIONS

The prototype is first validated by the PsdlBuilder while it is read from file. An error message will appear if a syntactic error is found in the PSDL file. The program will stop at this point. Thus, the PSDL Editor cannot modify or view a corrupted prototype file. As the PsdlBuilder successfully parses the prototype file, it constructs the data structures as well.

A second type of validation is performed when the PSDL code is created by the CreatePsdl routine. The PsdlParser is invoked with the created PSDL code. If the parsing of the code is successful, the prototype is saved in the file. If it is not, an error message is displayed.

The third type of syntactic validations takes place during the modifications of the operators and streams in the PSDL Editor. Each user input is validated by invoking the PsdlParser routines.

PsdlParser routine contains only static methods. An object of this class need not be
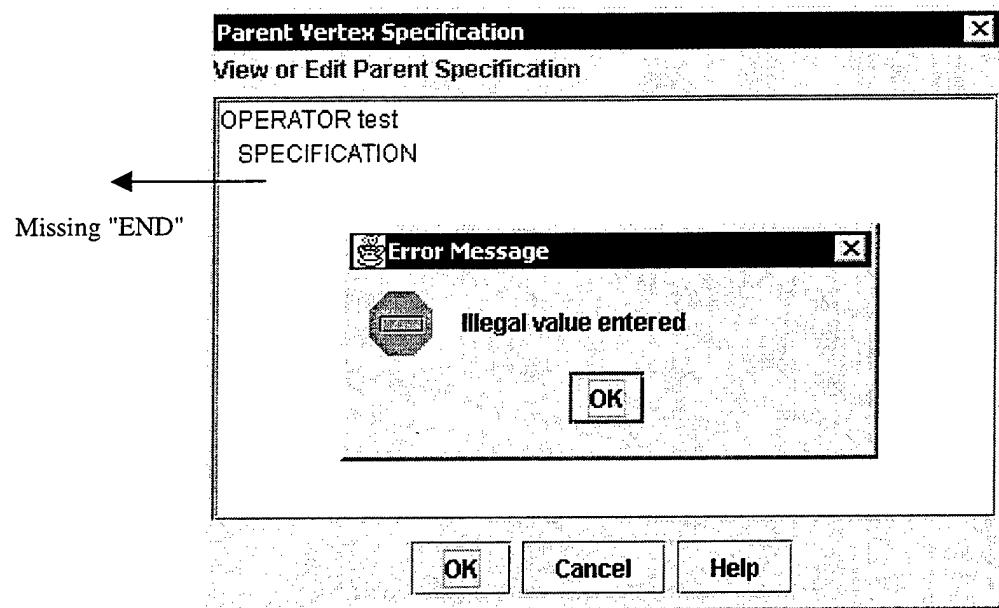
Figure 9. Syntactic Error Message

created to perform parsing. This allows partial parsing of PSDL grammar productions. GrammarCheck routine calls the necessary productions rules in PsdlParser. The input is accepted if the PsdlParser returns without throwing a ParseException. Otherwise, the user is informed that the input is not syntactically correct. Figure 9 shows an error message that is displayed when an attempt is made to acknowledge a syntactically incorrect "Parent Specification".

IV. USER INTERFACE IMPLEMENTATION

The main purpose of the PSDL Editor is to construct a PSDL prototype via a user friendly and easy to use graphical interface. The PSDL Editor of CAPS Release 2.0 managed to capture most of the PSDL constructs. Basic PSDL constructs could be directly synthesized from the Data Flow Diagram. Those Basic features of PSDL were enough for novice users to build executable prototypes. Users with more advanced knowledge of PSDL could use pop-up editors to create more complex constructs.

The graphical interface of this implementation is largely the same as CAPS Release 2.0. There are a few enhancements that make the use of the Editor easier. The Data Flow Diagram and their symbols are the same. However, manipulation of the data flow components on the diagram has changed slightly.

This implementation captures the PSDL constructs the same way as CAPS Release 2.0. Thus, users of previous versions will have no adaptation problems using this implementation.

A. PSDL EDITOR ENVIRONMENT

Almost all of the user interface components are implemented using Java Swing* components. The Swing library provides lightweight components, which execute more efficiently than the standard AWT components. Java Standard Look and Feel is used for the implementation, which gives a platform independent Look and Feel. Executing the program under Windows or Solaris will result in the same user interface. It was also possible to use Windows style or Motif style look and feel, but the main idea behind this research was to implement HSI in a platform independent way.

---

* Java and Swing are trademarks of Sun microsystems.

1. PSDL Editor Layout

The Graph Editor layout has been changed in this implementation with a few added features. However, the functionality of the Editor is the same: to allow the user to create the specifications of an operator and its data flow implementation. It is still not possible to view or edit source language implementation (e.g. Ada or TAE) of an operator from the PSDL Editor.

The Editor consists of six main components as shown in Figure 10. These components and their functionality are explained in the following sections.

a. Main Window

This window is a JFrame object that holds the other components inside. It can not be run as a stand-alone program and must be invoked from the HSI main window by selecting the Edit-PSDL menu. To execute the PSDL Editor main window, either a new prototype must be created or a parsable prototype file must be provided. If a new prototype is created, the main window will open with an empty prototype. If an existing prototype is selected, the main window will construct the data flow diagram that it parses from the prototype file. In either case, the name of the prototype file will appear on the Title Bar as the name of the current prototype.

The main window will be initialized as an 800x600 window, and it will be placed in the middle of the screen. Thus, it is not recommended to use the editor with a resolution less than the specified size. However, it is still possible to resize the window after it is constructed.

The main window can be closed by using the 'Exit' menu item in the 'File' menu, or by clicking on the 'close window' icon (☒) on the title bar. If the prototype has changed since the main window has opened, the program will prompt the user to save the prototype. The program will exit without saving the changes if 'no' is selected. The window closing operation can also be cancelled without changing the status of the prototype.

b. Menu Bar

The menu bar provides five pull down menus. Figure 11 depicts the menu
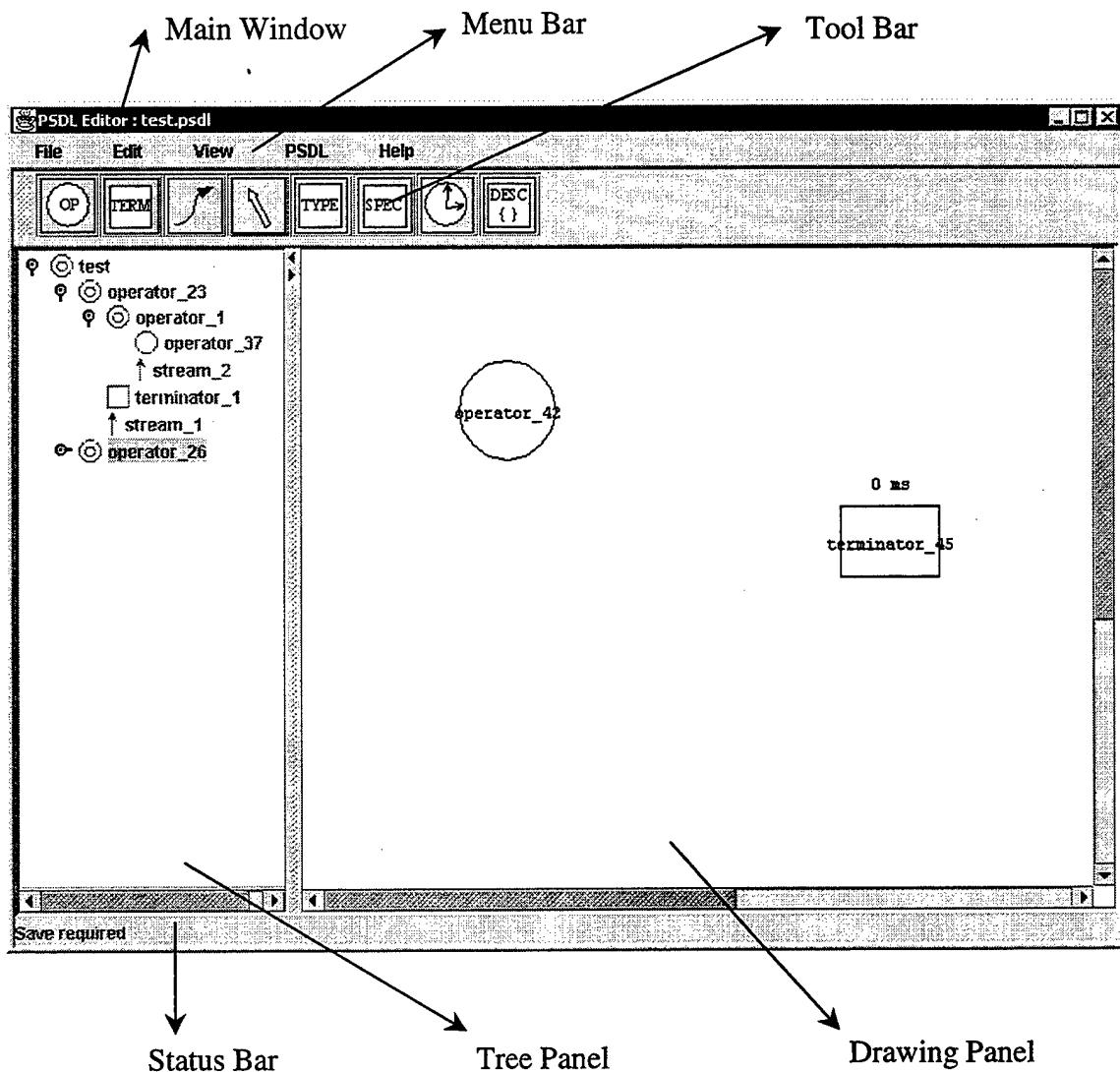
Figure 10. PSDL Editor Layout

bar and menu selection. The menus and the menu items are selected by using the left mouse button. Navigating through the menu items will also display the functionality of the menu item on the status bar.

Not all the menu items are available at the same time. If a menu item is not available for a specific situation, it will be grayed out and an attempt to use the item will have no effect. The functionality and availability of the menu items are displayed in Table 1.

21
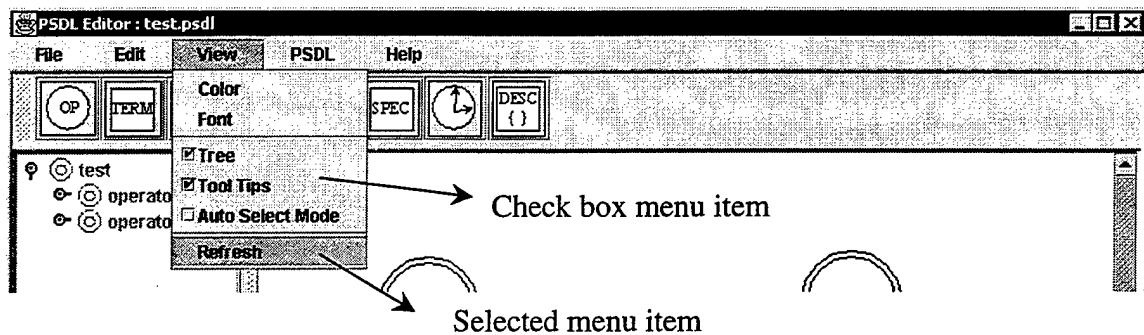
Figure 11. PSDL Editor Menu Bar and Menu Items

Some menu items are check box menu items. Selecting and deselecting the check box menu items will turn on and off the option.

c. Tool Bar

The Tool Bar is located under the Menu Bar and contains eight buttons. The first three of these buttons are used to insert data flow components into the drawing panel. An operator, a terminator or a stream will be placed into the drawing panel when the mouse is first pressed on the button, and then on the drawing panel. There are two ways that components can be placed into the drawing panel. If 'Auto Select Mode' is selected under the 'View' menu, the editor will enter into the Select mode after a component is placed. Otherwise, components will be placed into the drawing panel continuously, i.e., each time the mouse button is pressed on the drawing panel, a new component will be added to the data flow diagram.

The fourth button is used to select components from the data flow diagram. The functionalities of the remaining four buttons are described in Table 2. They open editor windows to view or modify the properties of the parent operator of the current data flow diagram. An exception to that is the Types button, which opens a text editor window to modify the properties of the global data types.

Tool Bar buttons also provide tool tips. Placing the mouse over a button will display the tool tip after two seconds. Tool tips can be enabled or disabled from the view menu.

Menu	Identification	Functionality	Availability
File	Save	Saves the current prototype to the disk	Not available if no changes are made to the prototype.
	Restore From Save	Ignores changes and restores the data flow diagram from the prototype file	Not available if no changes are made to the prototype
	Print	Prints the data flow diagram	Always
	Exit	Quits the PSDL Editor. Prompts for save.	Always
Edit	Undo	Undo for the last action.	Not available if no undoable action exists
	Redo	Redo for the last action.	Not available if no redoable action exists
	Select All	Selects all of the components currently on the draw panel.	Always
	Delete	Deletes the selected component	Not available if no component is selected from the draw panel
View	Color	Changes the current color of the draw panel	Always
	Font	Changes the current font of the draw panel	Always
	Tree	Hides or views the tree panel	Always

Table 1. Menu Items

Menu	Identification	Functionality	Availability
View	Tool Tips	Enables or disables the tool tips for the tool bar buttons.	Always
	Auto Select Mode	If not checked, component insertion to the draw panel will be continuous.	Always
	Refresh	Refreshes the components on the draw panel	Always
PSDL	Go to Root	Changes the level to the root operator	Not available if the current operator is the root operator.
	Go to Parent	Changes the level to the parent of the current operator.	Not available if the current operator is the root operator.
	Decompose	Decomposes the current operator	Not available if the selected component is a stream
Help	PSDL Grammar	Provides help for PSDL Grammar	Always
	Operators	Provides help for Operators	Always
	Streams	Provides help for Streams	Always
	Exceptions	Provides help for Exceptions	Always
	Timers	Provides help for timers	Always

Table 1. Menu Items

24

Button	Functionality
OP	Draws an operator into the drawing panel
TERM	Draws a terminator into the drawing panel
	Draws a stream into the drawing panel
	Selects a component from the drawing panel
TYPE	Opens text editor to view or edit Data Types
SPEC	Opens text editor to view or edit Parent Specs
	Opens id-list editor to view or edit Timers
DESC {}	Opens text editor to view or edit Informal Graph Description

Table 2. Toolbar Buttons

d. Tree Panel

One of the new features of this implementation is the Tree Panel that is placed to the left of the Drawing Panel (Refer to Figure 10). In CAPS Release 2.0, there was no global view of the PSDL prototype. The user had to traverse all intermediate nodes to modify a child operator that was five levels down. It was quite a time consuming operation when the prototype would be checked syntactically in every visited operator.

Tree Panel provides global view of the PSDL Data Flow Graph hierarchy. The root operator is placed at the top of the Tree Panel. The children of an operator are placed under their parent. Expanding a parent operator make it children visible in the Tree Panel while collapsing the parent operator will hide them.

Component	Icon
Composite operator	◎
Atomic operator	○
Composite terminator	▣
Atomic terminator	☐
Stream	↑
State stream	↑

Table 3. Tree Panel Icons

Selecting a component on the Tree Panel will have different effects according to its type. If the component is a stream, it will be the same as if the select button on the toolbar is pressed and then the stream itself is selected. It does not matter which level the current data flow diagram represents, the drawing panel will automatically change its level to the selected stream's level. If the selected component is an operator or a terminator, the effect will be the same as if the component is decomposed. If the operator is already composite, the children of the operator will be visible in the drawing panel.

If the operator is atomic, a blank data flow diagram will be opened to place new components as its children.

Each component is represented on the Tree Panel by both its name and an icon specifying the type of the component. Table 3 shows the icons that are used on the Tree Panel. The Tree Panel also supports horizontal and vertical scroll bars that are automatically placed when there is not enough space to represent the components.

The Tree Panel can be hidden to provide more space for the Drawing Panel. This action can be performed either by clicking on the small triangle icons that are on the separator or by de-selecting "Tree View" menu item from the "View" menu.

e. Drawing Panel

Data Flow Components are displayed in the Drawing Panel. The size of the drawing panel is fixed to 1024x768. Changing the size of the Main Window will have no effect on the size of the Drawing Panel. Both vertical and horizontal scroll bars are provided to access the unseen areas of the Drawing Panel.

f.   Status Bar

A status Bar is located in the lower part of the Main Window and provides feedback to the user. Two kinds of information are available from the status bar. According to whether or not the prototype has been modified, the status bar will display either "Save Required" or "Save Not Required".

The second kind of information is provided when the mouse is placed on a button in the tool bar or on a menu item. The functionality of the buttons and menu items are displayed as mini-help in the status bar. It is not necessary to press the buttons for the Status Bar to display their functionality.

2.   Cursor Types

There are three types of cursors that are used within the PSDL Editor. Hand Cursor is used when the editor is in the select mode and the mouse is over a data flow component on the display area. Hand cursor is also displayed when the mouse is over the label of a component, or met of an operator, or latency of a stream.

Move cursor is displayed during the relocation of a component or a label on the drawing panel. It is also used when an operator is resized. For all other purposes, the arrow shaped default cursor is used.

3.   Mouse Interface

The PSDL Editor assumes a two-button mouse. The right mouse button is only used to launch a pop-up menu that is used to change the properties of a component on the drawing panel. This is accomplished by placing the mouse over a component or a label (the cursor will change to hand-cursor at this point) and pressing the right-mouse button. For all other features, the left-mouse button is used.

Menu	Menu Item	Hot Key
File	Save	CTRL-S
	Print	CTRL-P
Edit	Undo	CTRL-Z
	Redo	CTRL-Y
	Select All	CTRL-A
	Delete	DEL
View	Refresh	CTRL-F
PSDL	Go to Root	CTRL-R
	Go to Parent	CTRL-O
	Decompose	CTRL-D

Table 4. PSDL Editor Hot Keys

4. Hot Keys

Hot keys provide quick access to most of the menu item functionalities. Hot keys are also identified in the menu items. Table 4 shows the hot keys and their associated menus and menu items.

B. DATA FLOW DIAGRAM

The implementation of composite operators is displayed in the drawing panel. Objects that are used to represent the data flow diagram components are the same as those that are used in CAPS Release 2.0. The PSDL Grammar has productions to support constructing a data flow diagram from a PSDL file.

Visual modification of components on the drawing panel is available to improve readability of the diagram. They include resizing objects, moving them from one place to another and changing colors and fonts. These are explained in the following sections.
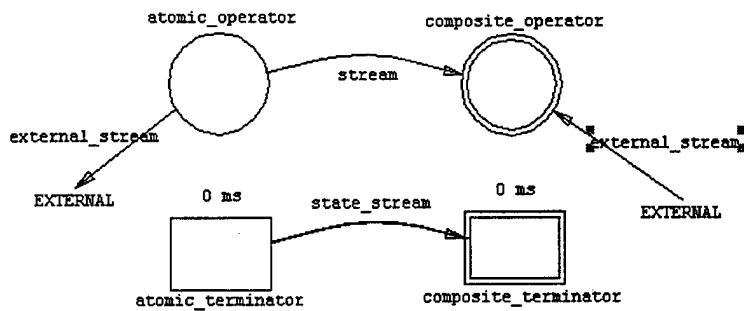
Figure 12. Data Flow Diagram Symbols

1. Data Flow Diagram Symbols

Figure 12 shows the symbols that are used to represent the data flow diagram components. Circles represent operators, rectangles represent terminators and lines represent streams.

PSDL data flow graph consists of a network of operators. An atomic operator is implemented by a supported programming language while a composite operator is implemented by a network of PSDL operators. Terminators are a special kind of operator. They represent the systems or subsystems that lie outside of the intended system. These systems or subsystems have a maximum execution time of zero since their processing time should be excluded from the total time spent on the intended system during prototype execution. Composite operators are represented by double circles and composite terminators are represented by double rectangles.

The distinction between streams and state streams is specified by using a thicker line for the state streams. The direction of data flow from source to the destination is indicated by an arrow.

Streams that enter or exit a composite operator are specified as EXTERNAL in the implementation of the operator. The source or the destination of the stream may be EXTERNAL, but not both.

2. Drawing Data Flow Components

29

For convenience to the users, there are two ways to draw components on the drawing panel. If the "Auto Select Mode" menu item is not checked from the "View" menu, the selected drawing tool from the tool bar will remain active until another one is selected. This provides CAPS Release 2.0-style placement of the components, as many of the selected components can be drawn without reselecting them from the Tool Bar.

In the second way, if the "Auto Select Mode" is checked, the PSDL Editor will enter into "Select Mode" as if the "Select" button from the Tool Bar is pressed after a component is drawn into the drawing panel. For successive drawings, the tools must be selected from the Tool Bar each time.

a.   Operators and Terminators

The procedures to draw or modify operators and terminators are the same. From now on, we will only describe drawing operators. The same procedures can be applied to the terminators as well.

An operator is drawn by first pressing and releasing the operator button from the Tool Bar and then performing a left-click over the desired location of the operator in the drawing panel.

The PSDL Editor remembers the current font and the current color. A new operator that is drawn into the drawing panel will be drawn in the current color of the editor. Similarly, labels (including met and latency) will be specified in the current font of the editor. These properties can be changed as described later.

Inserting a new operator or a new stream will automatically update the tree panel. The name of the operator or the stream will be visible under its parent operator.

b.   Streams

A stream consists of a set of control points. To draw a stream, the streams button form the Tool bar is clicked first. Control points are added with each left-click on the drawing panel. The first control point is the source of the stream. Performing a left-click when the cursor is over an operator will assign it as the source of the stream. Any number of intermediate control points can then be added to the stream's path by left clicking on the empty areas. The last control point is the destination of the stream. By
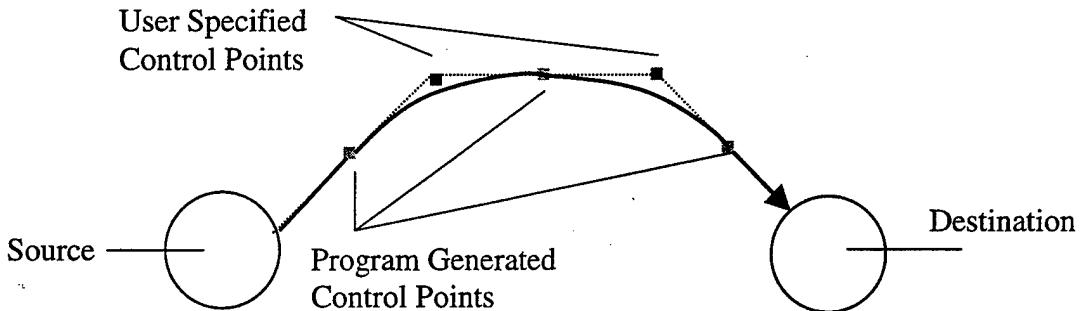
Figure 13. Construction of a stream

left-clicking over the destination operator, the drawing of the stream is complete. The stream will be drawn from the source to the destination with an arrowhead on the destination side.

Figure 13 shows how a stream is constructed. The stream is not actually drawn following the control points. Internal to the implementation, more control points are added to the path of the stream. The midpoint of two control points is added as a new control point. Insertion of the intermediate points provide a smooth curve.

For composite operators, input and output streams to the composite operators are specified as EXTERNAL in their implementation. Drawing a stream from EXTERNAL to another operator is the same except that the mouse is clicked on an empty area to specify EXTERNAL as the source. If the destination is EXTERNAL, the stream is constructed normally until the destination operator. Instead of selecting an operator as destination, the cursor is located on an empty area and right mouse button is clicked. This will draw an operator-to-EXTERNAL stream.

The insertion of a stream into the Data Flow Graph can be cancelled any time during the construction of the stream by hitting "Esc" from the keyboard. All of the intermediate control points will be deleted from the Drawing Panel.

c.   *Labels*

Labels represent the name of the component, the maximum execution time of operators and terminators and the latency of streams. They are placed relative to the

31
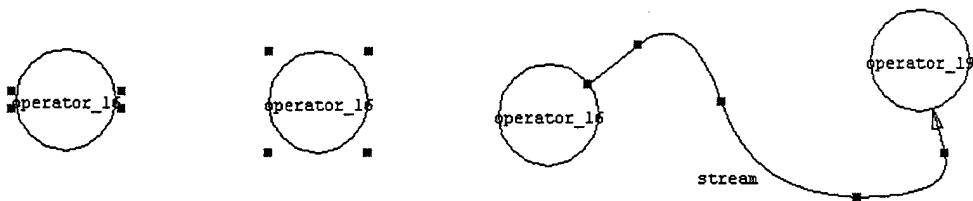
Figure 14. Selected Label, Operator and Stream

position of the component. For operators and terminators, labels are placed relative to the center of the component. For streams, labels are placed relative to the mid-control point.

3. Modifying the Data Flow Diagram

a. Selecting Components

Data Flow Components are selected from the drawing panel when the editor is in the Select mode. This can be provided either by clicking on the select button on the tool bar or by enabling "Auto Select mode" menu item from the "View" menu. The cursor will change to the hand cursor when the mouse is over the component. For streams, this will happen when the mouse is over or near one of the control points.

Another way to select a stream is to click on the name of the stream from the tree panel. This is a more convenient method especially when the stream is in another level of decomposition. The level of the graph will be changed automatically to the level of the selected stream.

Labels are selected the same way as the operators are selected. The cursor will change to hand cursor over the label and when selected, handle points will appear around the label.

Figure 14 shows a selected operator and a selected stream. Component handles are shown as small squares to notify the user that the component is selected.

b. Relocating Components

Components can be relocated on the drawing panel to improve the readability of the graph. To accomplish this, the editor must be in the Select mode and the

32

component that is to be relocated must be selected by clicking on the component. Component handles will be visible at this point as shown in Figure 14.

To move operators and terminators, the user must press the left mouse button over the component and drag the component to its new location. The drawing panel is continuously updated while dragging the component. Paths of input and output streams of the relocated component are altered while the component is dragged. The intermediate control points of the streams are not changed, only the end point towards the relocated component is updated.

Streams can not be completely moved in the drawing panel since they are tied at both ends by their source and destination. But stream paths can be changed. To do this, one of the control points must be dragged to a new location. Other control points will be updated to preserve the smoothness of the path.

Moving components will also carry their labels to their new relative location. The labels themselves can also be relocated to a new offset value. To do this, move the cursor over the label (the cursor will be changed to the hand cursor), press the left mouse button and drag the label to a new location.

We provided a new functionality in this implementation that would be useful when the user wants to place a component in an area where there is not enough space. In CAPS Release 2.0, the user had to move the components one by one, which would change the look of the graph differently than its original version. With this implementation, it is possible to select all of the components in the current level and move them to a new location. This is done by choosing "Select All" menu item from the "View" menu. The handles of all the components and labels will be visible at this point. Then, by pressing left mouse button over any component or label and dragging the mouse will relocate all the components to a new location. Left clicking the mouse in an open area of the drawing panel will de-select the components.

While moving components, it is not possible to go out of the area of the drawing panel. The component will stop on the border where it is still visible and will not move further with mouse movements. It will only move again when the mouse is dragged
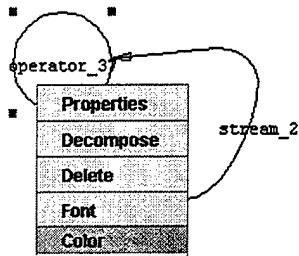
Figure 15. Component Pop-up Menu

towards the inside of the drawing panel. This functionality is also available when labels are moved by themselves. However, labels may go out of the visible area when a component is moved, causing the relative location of the label to go into the invisible area. When this happens, the user should first relocate the label to another side of the operator before moving the operator towards the border of the drawing panel.

c. Resizing Components

Only operators and terminators can be resized. The operator is selected first. Then, the mouse is placed over one of the handles and dragged to a new location.

The operator will be resized towards the direction of the dragging operation. The drawing panel is again continuously updated while resizing operators. Labels will move to a new location relative to the new center of the component. It is not possible to resize the component towards the invisible area of the drawing panel.

d. Using the Pop-up Menu

A pop-up menu will appear when the editor is in select mode and a right-click is performed while the mouse is over a component. This pop-up menu is the same for operators and streams except that decompose and color menu items are disabled for the streams. The pop-up menu will also appear with a right click over the associated labels of the components.

Figure 15 depicts the pop-up menu that is opened for an operator. Decompose, color, font and delete menu items have the same functionality as their corresponding menu items from the menu bar as explained in Table 1.
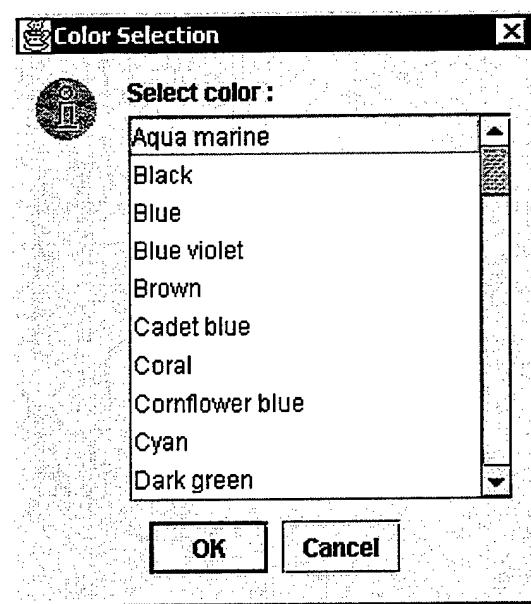
Figure 16. Color Selection Dialog

Selecting the properties menu item will launch the properties window for the component. The properties window and its functionality will be explained later in this chapter. It can also be launched by double-clicking with the left mouse button while the mouse is over a component or over the label of a component.

e. Deleting Components

Deleting procedure for operators and streams are the same. The component is selected first. It can then be deleted by using the "delete" menu item from the pop-up menu or from the "Edit" menu. It is also possible to delete the component by using the DELETE key form the keyboard. If a component is deleted from the drawing panel, it will also be deleted from the tree panel.

Deleting an operator will also delete its input and output streams. Deleting a composite operator will delete all of the sub-components of that operator.

e. Changing Colors

The fill colors of the operators can be changed to improve the readability of the diagram. This feature is not available for streams. The PSDL Editor maintains a current color value and it is initialized as the color white during startup. The value of the

35
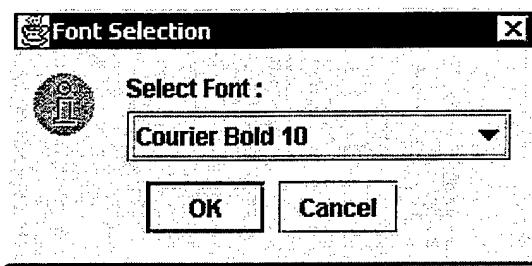
Figure 17. Font Selection Dialog

current color can be changed from the "Color" menu item which is in "View" menu. As shown in Figure 16, a dialog window will appear prompting the user to select a color value. Once a value is selected from this dialog window, it will remain as the current color until a new value is specified using the same method. All operators and terminators that are placed on the drawing panel are painted with the current color of the editor.

The color of an existing component can be changed by using the "Color" menu item from the pop-up menu. Right clicking on a component, as explained earlier, will launch the pop-up menu. The color dialog window as in Figure 16 will appear when "Color" menu item is selected. The color of the operator will be changed immediately to the color value that is selected from the dialog window. This operation is effective only for the selected component.

f. Changing Fonts

Fonts are used for the labels of the components. Changing fonts is similar to changing colors. Fonts can be specified to be the current font of the editor for future components. Again, it is possible to change the font of only the selected component. One difference is that changing fonts is also available for the streams. Figure 17 depicts the font selection dialog.

4. Navigating the Prototype

As mentioned earlier, a PSDL prototype consists of a network of operators. The PSDL Editor displays the data flow diagram of only one operator at a time. It is possible

36
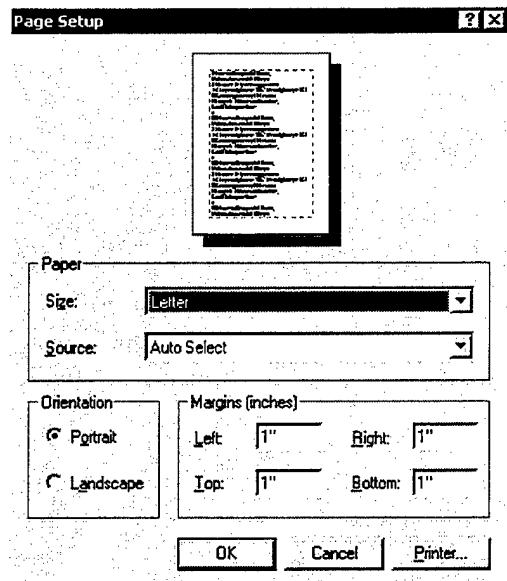
Figure 18. Print Dialog

to traverse and view the other levels of the hierarchy as well. There are four ways to navigate through the hierarchy.

The "PSDL" menu provides three menu items. "Go to Parent" and "Decompose" menu items will traverse the prototype one level up or one level down. "Go to Root" menu item is a shortcut to directly traverse to the root operator. All of these functionalities can also be invoked by Hot Keys as described in Table 4.

Another way to decompose an operator or a terminator is to use the "Decompose" menu item from the pop-up menu. The pop-up menu opens when a right-click is performed over the selected component.

The last and the most convenient way to traverse the prototype hierarchy is to use the tree panel. Selecting an operator from the tree panel will automatically decompose that operator and the children of the selected operator will be displayed in the drawing panel. If the selected operator has no children, the drawing panel will display a blank page. But, the operator will not become a composite operator until at least one component is placed as its child.

If a stream is selected from the tree panel, the level will be automatically changed to the level containing the stream, and the stream handles will be displayed in the drawing panel.

5.   Printing the Data Flow Diagram

Selecting the "Print" menu item from the "File" menu or using the associated Hot key will open a print dialog to select the orientation of the printout. Figure 18 shows the print dialog that opens on a Windows NT system. A similar dialog will open for other operating systems. It is possible to set the orientation to PORTRAIT or LANDSCAPE, but changing the page margins will have no effect in the printout. Each level of the prototype hierarchy is sent to the printer as a different page. At the top of the printed diagram, the name of the parent operator of that data flow diagram is also printed.

6.   File Operations

PSDL Editor provides only two kinds of file operations. Selecting the "Save" menu item under the "File" menu will save the prototype to the disk. This operation is not available when the status bar displays "Save not required". The editor is sensitive to the changes that are made to the prototype. If the prototype is modified, the status of the editor will be "Save required" and this will be displayed in the status bar. The actions that cause this situation include creating new components, moving components, resizing components, deleting components and changing the properties of components.

The other kind of file operation is the "Restore from save", which resets all the changes that are made to the prototype since it was last saved. The user will be prompted that all the changes will be lost before restoring the prototype. If the user acknowledges, the last saved version will be read from the disk and its prototype will be displayed.

If the prototype is modified, an attempt to close the editor without saving the prototype will launch a warning message. The user can save and exit the program, exit the program without saving or cancel the closing operation.

Vertex Properties

Name : operator_2   Operator ▼

Implementation Language : Ada ▼

Trigger :
By Some ▼   Stream List
If Condition
Required By

Timing :
Periodic ▼
MET : 12   ms ▼   Required By
Period : 13   ms ▼   Required By
Finish within : 15   ms ▼   Required By

Output Guards   Exception Guards
Exception List   Timer Ops

Keywords   Informal Desc   Formal Desc

OK   Cancel   Help
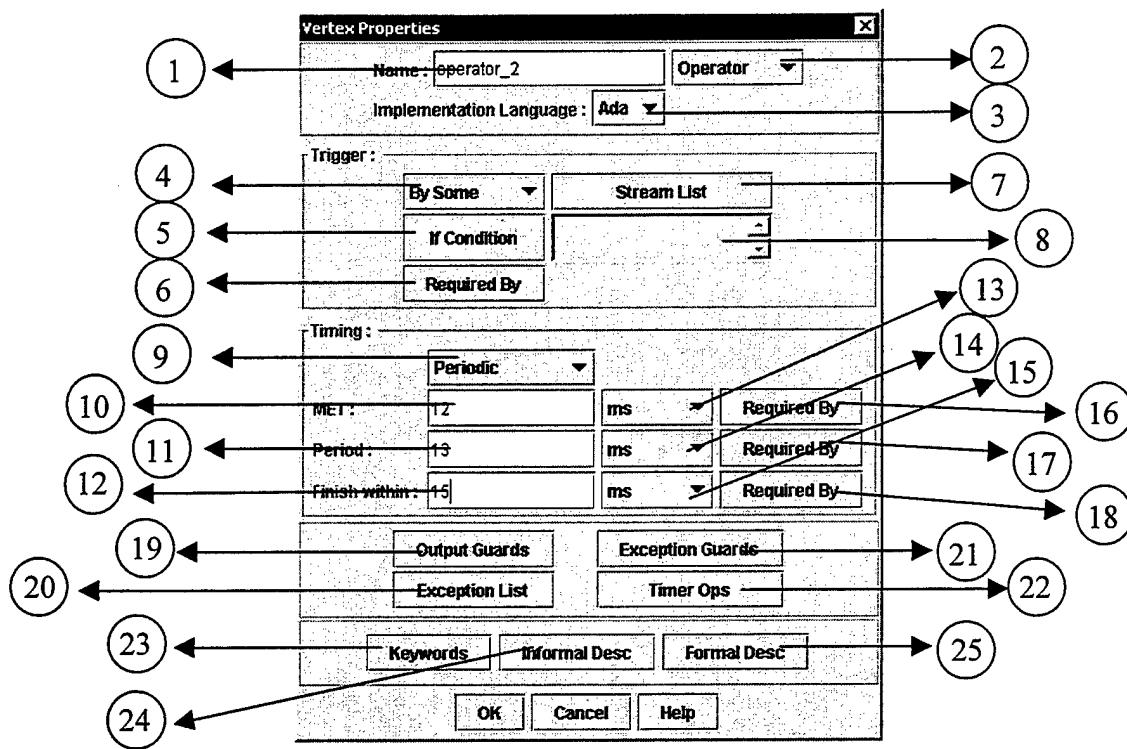
Figure 19. Operator Properties Dialog

StreamProperties

Stream Name :   unnamed_stream_5

Stream Type :   undefined_type

Is it a state stream ?   ○ No   ● Yes

State Initial Value   13

Latency :   ms ▼

OK   Cancel   Help
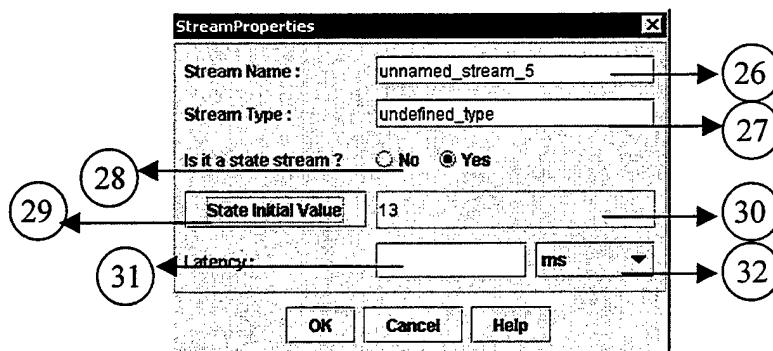
Figure 20. Stream Properties Dialog

OP   TERM   TYPE   SPEC   DESC { }
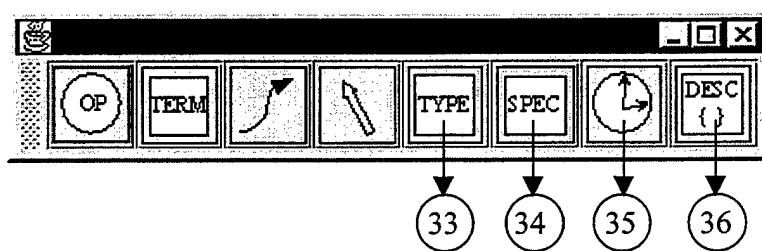
Figure 21. Tool Bar Buttons

Window	Index	Component name	Component	Component functionality	Validation rule
Operator	1	Operator name	Data entry	Operator name is entered into the text area	id
	2	Operator type	Combo box	Changes an operator to terminator or vice versa	n/a
	3	Implementation language	Combo box	Changes implementation language to Ada or Tae	n/a
	4	Trigger type	Combo box	Unprotected, by some or by all	n/a
	5	Trigger if condition	Text editor	Opens text editor to edit trigger if condition	expression
	6	Trigger required by	Id-list editor	Opens id-list editor to edit trigger requirements	id_list
	7	Trigger identifier list	Id-list editor	Opens id-list editor to edit trigger streams list	id_list
	8	Trigger if condition exp.	Display only	If condition exp. is displayed in the text area	n/a
	9	Timing type	Combo box	Not-time-critical, periodic or sporadic	n/a
	10	MET Value	Data entry	MET value is entered into the text area	integer_literal
	11	Period/MCP Value	Data entry	Period or MCP value is entered into the text area	integer_literal
	12	Finish within/MRT value	Data entry	Finish within/MRT value is entered here	integer_literal
	13	MET units	Combo box	Microsec, ms, sec, min or hours	n/a
	14	Period/MCP units	Combo box	Microsec, ms, sec, min or hours	n/a
	15	Finish within/MRT units	Combo box	Microsec, ms, sec, min or hours	n/a
	16	MET required by	Id-list editor	Opens id-list editor to edit met requirements	id_list
	17	Period/MCP required by	Id-list editor	Opens id-list editor to edit period requirements	id_list
	18	Finish within/MRT req.by	Id-list editor	Opens id-list editor to edit FW-MRT requirement	id_list

Table 5. Component Properties Dialog Functionality

Window	Index	Component name	Component	Component functionality	Validation rule
Operator	19	Output guards	Text editor	Opens text editor to edit operator output guards	check_output_guards
	20	Exception list	Text editor	Opens text editor to edit operator exceptions	check_exception_list
	21	Exception guards	Text editor	Opens text editor to edit operator output guards	check_exception_guards
	22	Timer ops	Id-list editor	Opens id-list editor to edit operator timer-ops	check_timer_ops
	23	Keywords	Id-list editor	Opens id-list editor to edit operator keywords	id_list
	24	Informal description	Text editor	Opens text editor to edit operator informal desc.	check_informal_desc
	25	Formal Description	Text editor	Opens text editor to edit operator formal desc.	check_formal_desc
Stream	26	Stream name	Data entry	Stream name is entered into the text area	id_list
	27	Stream type	Data entry	Stream type is entered into the text area	type_name
	28	State stream selection	Radio button	Sets the stream type to state stream or not	n/a
	29	State stream initial value	Text editor	Opens text editor to edit state stream initial value	initial_expression
	30	State initial value display	Display only	Shows the state initial value	n/a
	31	Latency Value	Data entry	Latency value is entered into the text area	integer_literal
	32	Latency units	Combo box	Microsec, ms, sec, min or hours	n/a
Tool Bar	33	Data Types	Text editor	Opens text editor to edit data types	psdl
	34	Parent Specifications	Text editor	Opens text editor to edit parent specifications	check_parent_spec
	35	Timers	Id-list editor	Opens id-list editor to edit timers	id_list
	36	Graph Description	Text editor	Opens text editor to edit graph description	check_informal_desc

Table 5. Component Properties Dialog Functionality

C.   CHANGING COMPONENT PROPERTIES

It is possible to change operator and stream properties by using a dialog window. This dialog opens when the left mouse button is double clicked over a component or when the "Properties" menu item is selected from the pop-up menu. Different property dialogs are available for operators and streams. Figure 19 and Figure 20 show examples of these dialogs. Some of the operator properties can be modified using four buttons that are located in the Tool Bar. Figure 21 shows these buttons. The components of the dialogs and the Tool Bar are indexed in the figures. Refer to Table 5 for their functionalities. The following sections describe the types of components that are used to edit PSDL component properties.

1.   Display Only

These are the text areas that are disabled for user input and only display data values. State initial value (index 29) is an example of the display only text areas.

2.   Data Entry

Data entry areas accept user input from the keyboard. It may be necessary to click on the text area before entering data. If the data entered is longer that the provided area, it is not possible to see the whole data at once. However it is possible to scroll to the end of the text area by using the mouse or the left and right arrow keys. The entries or modifications to a data entry area will only be accepted when the user hits the "Ok" button on the window that contains the data entry area.

3.   Combo Boxes and Radio Buttons

Combo boxes and radio buttons provide a selection from provided choices. Only one of the choices can be selected at a time. Combo boxes have pull down menus to select the value. Triggering type selection (Index 4) is an example of the combo boxes. State stream selection buttons (Index 28) is an example of the radio buttons.

42
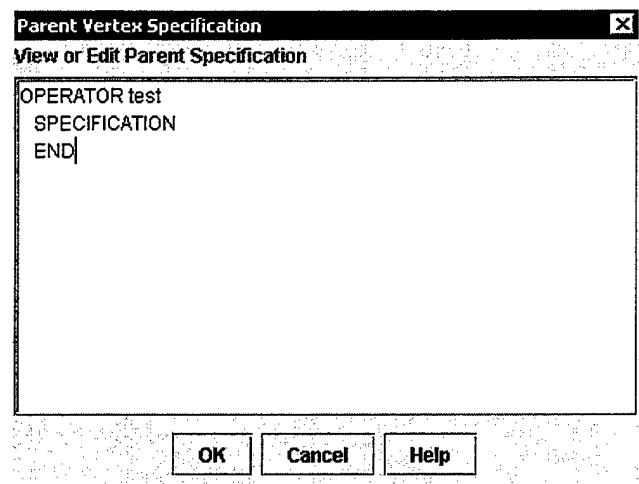
Figure 22. Text Editor
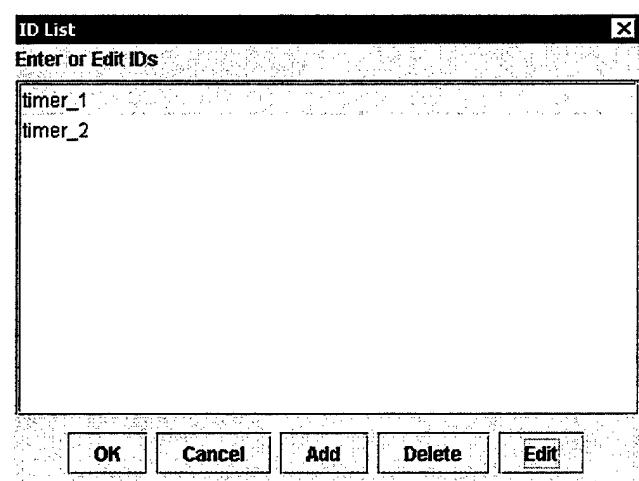
Figure 23. Id-list Editor
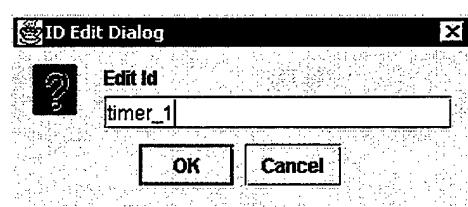
Figure 24. Id Edit Dialog

43

4.   Text Editor

Text editor is a dialog window that is used to view or edit the properties of PSDL components. The user can scroll through the text within the editor by using the arrow keys or by positioning the mouse cursor. Figure 22 shows a text editor that displays the specification of an operator. This is launched by pressing the "Parent Specs" button on the tool bar (Index 34).

The user can make modifications in the text editor. These modifications will not be reflected to the prototype until the "OK" button is depressed. The modifications can be aborted by depressing the "CANCEL" button any time.

5.   Id-list Editor

This dialog window is used to view or modify identifier lists. The modifications are not done directly by using keyboard on this editor. It views the identifiers as a list. Figure 23 shows an id-list editor that is accessed by clicking on the "Timers" button on the Tool bar (Index 35).

A new identifier can be added by pressing the "Add" button in the editor. A small dialog will open to enter a new identifier. This identifier is added to the list by pressing the "OK" button. Similarly, existing identifiers can be edited by pressing "Edit" button. A small dialog will open to edit the identifier. Figure 24 depicts the dialog that is used to edit the identifier "timer_1". If the "Edit" button is depressed when no selection is made from the id-list, a warning message will appear on the screen.

Existing identifiers can be deleted from the list by depressing the "Delete" button. Again, a warning message will be displayed when the user attempts for a delete operation without selecting an id first.

Similar to the text editor, the changes will be local to the id-list editor unless the user accepts them by depressing the "OK" button. All of the changes may be aborted by using the "Cancel" button.

V. CONCLUSION

A. RESULTS OF THIS RESEARCH

This research effort has resulted in a portable PSDL Editor for the Heterogeneous Systems Integrator. It is now possible to create PSDL prototypes on any machine and operating system where a Jdk1.2 compatible runtime environment can be found.

The performance of the program may depend on the underlying hardware that it is executed. We executed the program on a Pentium 120 MHz computer with 48 Mb RAM and on a Pentium II 400 MHz computer with 256 Mb RAM. While there was a significant difference in the performance of the two platforms, it was still possible create a prototype on the Pentium 120 MHz machine with a tolerable slowness in speed.

Basically, creating prototypes, displaying them on the user interface and modifying them are the same as CAPS Release 2.0 implementation. Users of previous CAPS versions will easily adapt to the new interface. There are some new features in this implementation, which do not affect the functionality of the program, but provide a friendlier interface and easier use.

The most powerful improvement that comes with this implementation is the addition of the Tree Panel to the Editor. The tree panel serves a few functionalities at the same time. It provides a better view of the overall prototype structure since all of the PSDL components can be seen in a hierarchy. The user can navigate through the prototype by clicking on the names of the components on the tree panel. Thus, it is possible to jump to any level in the hierarchy, which was not possible earlier.

The whole prototype is only checked syntactically when the user wants to save the prototype to the disk. In the previous implementation, some actions, such as navigating up or down through the prototype invoked a syntactic check on the prototype and then saved the prototype to the disk. If the user wanted to visit a component that was five levels down in the hierarchy, he/she had wait in all intermediate levels. These actions

45

are removed with this implementation. Moreover, changes in one level will be propagated to the other affected levels when the user visits another level of the prototype.

The CAPS 2.0 PSDL Editor maintained two copies of the prototype data structure in the previous version. The Background Checker maintained one of the data structures while the Editor maintained the other. This was necessary, because the implementation languages of the Background Checker and the PSDL Editor were different. Synchronization of the data structures was necessary, which was degraded the performance of the program. The new implementation maintains only one copy of the data structures. The root operator is passed as an argument to the utilities such as the parser. The entire data structure can be reconstructed by getting the children of the root operator.

New features have been added to the user interface such as selecting all components, moving all components on the drawing panel, and tool tips for the buttons of the Tool Bar. These features were explained in the previous chapters.

We provided HTML documentation of the source code in Appendix C. The documentation was created automatically using Javadoc. It documents the interface of the classes (i.e., the methods and the fields of the classes) without struggling through the implementation. We also provided small definitions of the fields and the methods of the classes in the documentation. This documentation will ease the task of program understanding in future evolutions of the PSDL Editor software.

B.   CRITICISMS OF THIS RESEARCH

Drawing of the components on the drawing panel is not very good. In particular, the circumference of the operators is not very smooth. Even though we used double precision classes, this problem was not solved. This problem may be due to the use of Swing components in the interface and if so, may be resolved with future releases of the Swing classes.

The PSDL Editor maintains a vector of the data flow components that are children of the current operator. While painting components on the screen, the program reads

them from this vector and draws them into the drawing panel. When a component is dragged or resized in the drawing panel, all of the components are first cleared and then repainted on the screen. This extra delay is normally not very significant. But it may get significant on a slower machine, on a busy processor or if the program is used over a network.

C.   RECOMMENDATIONS

With the previous versions of CAPS and this version of HSI, the user gets a feeling as if two separate programs are executing, the main window and the PSDL Editor. In a future implementation, these two seemingly separate programs can be incorporated into one user interface. The main program would include PSDL Editor as an internal frame in itself. Even more than one editor can be launched as internal frames inside the main window. This kind of an implementation will have a more modern look and feel than it has now.

The directory structure for the prototypes of CAPS and HSI depends on environment variables. A default directory to save and open prototypes is provided in both implementations. However, it would be better in my opinion, to leave the choice of the prototype directory to the user. Like some current Integrated Development Environments, the main program could open a project file, which contains all the information such as the locations of the prototype files, the version number, etc.

We had to give up many good ideas to provide backward compatibility with CAPS Release 2.0. The current version supports 64 colors and 6 fonts for the display of the components. These colors and fonts are indexed and do not allow any expansion. The structure of PSDL grammar need not change to support more fonts and colors since an expression is sufficient to provide a color or font value (Appendix A, grammar rule 23). The color values can be saved into the prototype as the integer RGB values. Fonts can be saved as string literals. This way, CAPS and HSI programs can support more colors and fonts.

APPENDIX A. PSDL GRAMMAR*

The following is the complete specification of the Prototype System Description Language (PSDL) syntax extended Backus-Naur Form (BNF).

The BNF description of PSDL specifies the sequence of symbols, which consolidate a valid PSDL prototype. BNF describes the language in terms of production rules. Each production rule equates a non-terminal symbol to a sequence of terminal and non-terminal symbols. Terminal symbols are symbols, which can occur in PSDL. Non-terminal symbols are metalinguistic variables whose value is a sequence of symbols, which represent a PSDL construct.

Terminals are represented as **bold** symbols. Non-terminals are enclosed in angle brackets, < and >. Additional metasymbols are introduced in the extension of BNF to reduce the number of productions and non-terminals. These metasymbols are defined as:

- Square brackets, [ ] , to enclose optional items.
- Curly braces, { } , to enclose items which may appear zero or more times.
- Vertical bars, | , to represent a choice between items.
- Parentheses, ( ) , to represent a grouping of items.

In some cases, the metasymbols are also used as terminals within PSDL. In order to avoid confusion, such terminal symbols are enclosed within single quotes.

For ease of reference, each production rule is numbered on the left hand side. These numbers are not part of the PSDL syntax.

49

1. < psdl >

    ::= { < component > }

2. < component >

    ::= < data_type >

    | < operator >

3. < data_type >

    ::= **type** < id > < type_spec > < type_impl >

4. < type_spec >

    ::= **specification** [**generic** < type_decl >] [< type_decl >

    { **operator** < op_name > < operator_spec > }

    [< functionality >] **end**

5. < operator >

    ::= **operator** < op_name > < operator_spec > < operator_impl >

6   < operator_spec >

    ::= **specification** { < interface > } [< functionality >] **end**

7. < interface >

    ::= < attribute > [< reqmts_trace >]

8. < attribute >

    ::= **generic** < type_decl >

    | **input** < type_decl >

    | **output** < type_decl >

    | **states** < type_decl > **initially** < initial_expression_list >

50

| **exceptions** <id_list>
| **maximum execution time** <time>

9. < type_decl >
    ::=   < id_list > : < type_name > { , < id_list > : < type_name > }

10. < type_name >
    ::=   < id >
    | < id > ' [' < type_decl > '] '

11. <id_list >
    ::=   < id > { , < id > }

12. < reqmts_trace >
    ::=   **required by** < id_list >

13. < functionality >
    ::=   [< keywords >] [< informal_desc >] [< formal_desc >]

14. <keywords >
    ::=   **keywords** < id_list >

15. < informal_desc >
    ::=   **description** ' { ' < text > ' } '

16. < formal_desc >
    ::=   **axioms** ' { ' < text > ' } '

17. < type_impl >
    ::=   **implementation** < id > < id > **end**

51

| implementation < type_name >
        { **operator** < op_name > < operator_impl > }   **end**

18.   < operator_impl >

       ::= **implementation** < id > < id > **end**

       | **implementation** < psdl_impl > **end**

19.   < psdl_impl >

       ::= < data_flow_diagram > [< streams >] [< timers >]

            [< control_constraints >] [< informal_desc >]

20.   < data_flow_diagram >

       ::= **graph** { < vertex > } { < edge > }

21.   < vertex >

       ::= **vertex** < op_id > [: < time >] { < property > }

22.   < edge >

       ::= **edge** < id > [: < time >] < op_id > —> < op_id > { < property > }

23.   < property >

       ::= **property** < id > = < expression >

24.   < op_id >

       ::= [< id > .] < op_name > ['(' [< id_list >] ' | ' [< id_list >] ') ']

25.   < streams >

       ::= **data_stream** < type_decl >

26. < timers >

      ::= **timer** < id_list >

27. < control_constraints >

      ::= **control constraints** < constraint > { < constraint > }

28. < constraint >

      ::= **operator** < op_id >

          [**triggered** [<trigger>] [**if** <expression>] [<reqmts_trace>]]

          [**period** < time > [< reqmts_trace >]]

          [**finish within** < time > [< reqmts_trace >]]

          [**minimum calling period** < time > [< reqmts_trace >]]

          [**maximum response time** < time > [< reqmts_trace >]]

          { < constraint_options > }

29. < constraint_options >

      ::= **output** < id_list > **if** < expression > [< reqmts_trace >]

      | **exception** < id > [**if** <expression>] [< reqmts_trace >]

      | < timer_op > < id > [**if** < expression >] [< reqmts_trace >]

30. < trigger >

      ::= **by all** < id_list >

      | **by some** < id_list >

31. < timer_op >

      ::= **reset timer**

      | **start timer**

      | **stop timer**

32. < initial_expression_list >

```
              ::=   < initial_expression > {  , < initial_expression > }


33.     < initial_expression >

        ::=   true

        |     false

        |     < integer_literal >

        |     < real_literal >

        |     < string_literal >

        |     < id >

        |     < type_name > . < op_name > [ '(' <initial_expression_list> ')' ]

        |     ' ( ' < initial_expression > ' ) '

        |     < initial_expression > < binary_op > < initial_expression >

        |     < unary_op > < initial_expression >


34.     < binary_op >

        ::=   and   |   or   |   xor

        |     <   |   >   |   =   |   >=   |   <=   |   /=

        |     +   |   -   |   &   |   *   |   /   |   mod   |   rem   |   **


35.     < unary_op >

        ::=   not   |   abs   |   -   |   +


36.     < time >

        ::=   < integer_literal >  < unit >


37.     < unit >

        ::=   microsec   |   ms   |   sec   |   min   |   hours
```

38. < expression_list >

::= < expression > { , < expression > }

39. <expression_list >

::= **true**

| **false**

| < integer_literal >

| < time >

| < real_literal >

| < string_literal >

| < id >

| < type_name > . < op_name > [' (' < expression_list > ') ']

| ' (' < expression > ') '

| < expression > < binary_op > < expression >

| < unary_op > < expression >

40. < op_name >

::= < id >

41. < id >

::= < letter > { < alpha_numeric > }

42. < real_literal >

::= < integer_literal > . < integer_literal >

43. < integer_literal >

::= < digit > { < digit > }

44. < string_literal >

    ::=   " { < char > } "

45. < char >

    ::=   any printable character except ' } '

46. < digit >

    ::=   **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**

47. < letter >

    ::=   **a | b | c | d | e | f | g | h | i | j | k | l**

    | m | n | o | p | q | r | s | t | u | v | w | x

    | y | z | A | B | C | D | E | F | G | H | I | J

    | K | L | M | N | O | P | Q | R | S | T | U

    | V | W | X | Y | Z

48. < alpha_numeric >

    ::=   < letter >

    | < digit >

    | ' _ '

49. < text >

    ::=   { < char > }

APPENDIX B. PSDLParser and PSDLBuilder

PSDLParser and PSDLBuilder files contain JavaCC source code. JavaCC is a parser generator that is used with Java applications. It reads a grammar specification and converts it to a Java program that can recognize matches to the grammar.

JavaCC is available for download from the Internet. More information about JavaCC is available at www.suntest.com/JavaCC.

PSDLParser routines read an input stream that contains PSDL constructs. If it can successfully parse the stream, it will return without an exception. If any of the PSDL grammar constructs are violated, it will throw ParseException.

PSDLBuilder is similar to PSDLParser, but it also contains actions to create data structures to represent the prototype in PSDL Editor. These actions are embedded Java code. JavaCC automatically inserts these actions into the parser that it creates.

```
/*
    Program : PsdlGrammar.jj
    Author  : Ilker Duranlioglu
    This grammar file is written in JavaCC version 0.8pre1.
*/

options {
    IGNORE_CASE = true;          /* PSDL is not case sensitive                  */
    DEBUG_PARSER = true;         /* Set this flag to true to trace the parser calls */
}


PARSER_BEGIN (PsdlParser)

package caps.Parser;

import java.io.*;
import caps.Psdl.*;
import java.util.Vector;

public class PsdlParser {

    public static void main (String args[]) throws ParseException {
    }

} // End of the class PsdlParser

PARSER_END (PsdlParser)


/* White Space */
SKIP :
{
    " "
|   "\r"
|   "\t"
|   "\n"
}

/* Reserved Words */
TOKEN :
{
    < IF : "if" >
|   < MS : "ms" >
|   < SEC : "sec" >
|   < END : "end" >
|   < MIN : "min" >
|   < TYPE : "type" >
|   < EDGE : "edge" >
|   < TRUE : "true" >
|   < FALSE : "false" >
|   < GRAPH : "graph" >
|   < TIMER : "timer" >
|   < HOURS : "hours" >
|   < INPUT : "input" >
|   < PERIOD : "period" >
|   < STATES : "states" >
|   < AXIOMS : "axioms" >
|   < OUTPUT : "output" >
|   < VERTEX : "vertex" >
|   < GENERIC : "generic" >
|   < MICROSEC : "microsec" >
|   < OPERATOR : "operator" >
|   < KEYWORDS : "keywords" >
|   < PROPERTY : "property" >
|   < TRIGGERED : "triggered" >
|   < EXCEPTION : "exception" >
|   < INITIALLY : "initially" >
|   < EXCEPTIONS : "exceptions" >
|   < DESCRIPTION : "description" >
|   < SPECIFICATION : "specification" >
|   < IMPLEMENTATION : "implementation" >
}

/* Operators */
TOKEN :
{
/* Binary Operators */
    < OR : "or" >
|   < AND : "and" >
|   < MOD : "mod" >
|   < REM : "rem" >
|   < XOR : "xor" >
|   < GREATER_THAN : ">" >
|   < LESS_THAN : "<" >
|   < EQUALS : "=" >
|   < GREATER_OR_EQUAL_TO : ">=" >
|   < LESS_OR_EQUAL_TO : "<=" >
|   < DIVIDE_EQUALS : "/=" >
|   < PLUS : "+" >
|   < MINUS : "-" >
|   < AMPERCENT : "&" >
|   < STAR : "*" >
|   < FACTOR : "/" >
|   < STAR_STAR : "**" >
```

```
/* Unary Operators */
  | < ABS : "abs" >
  | < NOT : "not" >
}

/* String real, and integer literals */
TOKEN :
{
    < TEXT : "{" ( < CHAR_TEXT > )* "}" >
  | < #CHAR_TEXT : ~["}"] >

  | < STRING_LITERAL : "\"" ( < CHAR_LIT > )* "\"" >
  | < #CHAR_LIT : ~["}", "\""] >

  | < REAL_LITERAL : < INTEGER_LITERAL > "." < INTEGER_LITERAL > >

  | < INTEGER_LITERAL : < INT_DIGIT > ( < INT_DIGIT > )* >
  | < #INT_DIGIT : ["0" - "9"] >
}

/* Identifiers */
TOKEN :
{
    < IDENTIFIER : < ID_LETTER > ( < ID_LETTER > | < ID_DIGIT > | "_" )* >
  | < #ID_LETTER : ["a" - "z"] | ["A" - "Z"] >
  | < #ID_DIGIT : ["0" - "9"] >
}

/* Digits and letters */
TOKEN :
{
    < DIGIT : ["0" - "9"] >
  | < LETTER : ["a" - "z"] | ["A" - "Z"]>
}


/**
  * Production 1
  */
void psdl () :
{}
{
    ( component () )*
}

/**
  *  Production 2
  */
void component () :
{}
{
    data_type ()
  | operator ()
}

/**
  * Production 3
  */
void data_type () :
{}
{
    <TYPE> id () type_spec () type_impl ()
}

/**
  * Production 4
  */
/* <functionality> is directly included in this production */
void type_spec() :
{}
{
    <SPECIFICATION> [ <GENERIC> type_decl () ] [ type_decl () ]
    ( <OPERATOR> op_name () operator_spec () )*
    [ keywords () ] [informal_desc () ] [ formal_desc () ] <END>
}

/**
  * Production 5
  */
void operator () :
{}
{
    <OPERATOR> op_name () operator_spec () operator_impl ()
}

/**
  * Production 6
  */
/* <functionality> is directly included in this production */
void operator_spec () :
{}
{
    <SPECIFICATION> ( inter_face () )* [ keywords () ] [informal_desc () ] [ formal_desc () ] <END>
}
```

59

```
/**
 * Production 7
 */
void inter_face () :
()
{
    attribute () [ reqmts_trace () ]
}


/**
 * Production 8
 */
void attribute () :
()
{
    <GENERIC> type_decl ()
  | <INPUT> type_decl ()
  | <OUTPUT> type_decl ()
  | <STATES> type_decl () <INITIALLY> initial_expression_list ()
  | <EXCEPTIONS> id_list ()
  | "maximum execution time" time ()
}


/**
 * Production 9
 */
void type_decl () :
()
{
    id_list () ":" type_name () ( ","  id_list () ":" type_name () )*
}


/**
 * Production 10
 */
/* This production is mofidied to remove common prefix id () */
/*
void type_name () :
()
{
    id ()
  | id () "[" type_decl () "]"
}
*/

void type_name () :
()
{
    id () type_name_suffix ()
}

/** This production is to remove the common prefix id () */
void type_name_suffix () :
()
{
    "[" type_decl () "]"
  | empty_string ()
}


/**
 * Production 11
 */
void id_list () :
()
{
    id () ( "," id () )*
}


/**
 * Production 12
 */
void reqmts_trace () :
()
{
    "required by" id_list ()
}


/**
 * Production 13
 */
/*  This production is included directly in other productions,
    because it caused empty string
void functionality () :
()
{
    [ keywords () ] [informal_desc () ] [ formal_desc () ]
}
*/


/**
 * Production 14
 */
void keywords () :
()
{
```

```
   <KEYWORDS> id_list ()
}

/**
 * Production 15
 */
void informal_desc () :
{}
{
   <DESCRIPTION> < TEXT >
}

/**
 * Production 16
 */
void formal_desc () :
{}
{
   <AXIOMS> < TEXT >
}

/**
 * Production 17
 */
/* This production is causing a common prefix problem and is modified
void type_impl () :
{}
{
   <IMPLEMENTATION> id () id () <END>
 | <IMPLEMENTATION> type_name ()
   ( <OPERATOR> op_name () operator_impl () )* <END>
}
*/

void type_impl () :
{}
{
   <IMPLEMENTATION> id () type_impl_suffix ()
}

/** This production is to remove the common prefix "implementation" */
void type_impl_suffix () :
{}
{
   id () <END>
 | [ "[" type_name_suffix () "]" ] ( <OPERATOR> op_name () operator_impl () )* <END>
}

/**
 * Production 18
 */
/* This production causes a common prefix problem and hence is modified
void operator_impl () :
{}
{
   <IMPLEMENTATION> id () id () <END>
 | <IMPLEMENTATION> psdl_impl () <END>
}
*/

void operator_impl () :
{}
{
   <IMPLEMENTATION> operator_impl_suffix ()
}

/** This production is to remove common prefix "implementation" */
void operator_impl_suffix () :
{}
{
   id () id () <END>
 | psdl_impl () <END>
}

/**
 * Production 19
 */
void psdl_impl () :
{}
{
   data_flow_diagram () [ streams () ][ timers () ]
   [ control_constraints () ] [ informal_desc () ]
}

/**
 * Production 20
 */
void data_flow_diagram () :
{}
{
   <GRAPH> ( vertex () )* ( edge () )*
}

/**
 * Production 21
```

```
    */
void vertex () :
{}
{
    <VERTEX> op_id () [ ":" time () ] ( property () )*
}


/**
  * Production 22
  */
void edge () :
{}
{
    <EDGE> id () [ ":" time () ] op_id () "->" op_id () ( property () )*
}


/**
  * Production 23
  */
void property () :
{}
{
    <PROPERTY> id () "=" expression ()
}


/**
  * Production 24
  */
/* This production has common prefix problem and is modified
void op_id () :
{}
{
    [ id () "." ] op_name () [ "(" [ id_list () ] "|" [ id_list () ] ")" ]
}
*/

void op_id () :
{}
{
    op_name () [ "." id () ] [ "(" [ id_list () ] "|" [ id_list () ] ")" ]
}


/**
  * Production 25
  */
void streams () :
{}
{
    "data stream" type_decl ()
}


/**
  * Production 26
  */
void timers () :
{}
{
    <TIMER> id_list ()
}


/**
  * Production 27
  */
void control_constraints () :
{}
{
    "control constraints" constraint () ( constraint () )*
}


/**
  * Production 28
  */
void constraint () :
{}
{
    <OPERATOR> op_id ()
    [ <TRIGGERED> [ trigger () ] [ <IF> expression () ] [ reqmts_trace () ] ]
    [ <PERIOD> time () [ reqmts_trace () ] ]
    [ "finish within" time () [ reqmts_trace () ] ]
    [ "minimum calling period" time () [ reqmts_trace () ] ]
    [ "maximum response time" time () [ reqmts_trace () ] ]
    ( constraint_options () )*
}


/**
  * Production 29
  */
void constraint_options () :
{}
{
    <OUTPUT> id_list () <IF> expression () [ reqmts_trace () ]
  | <EXCEPTION> id () [ <IF> expression () ] [ reqmts_trace () ]
  | timer_op () id () [ <IF> expression () ] [ reqmts_trace () ]
}
```

```
/**
 * Production 30
 */
void trigger () :
{}
{
    "by all" id_list ()
  | "by some" id_list ()
}


/**
 * Production 31
 */
void timer_op () :
{}
{
    "reset timer"
  | "start timer"
  | "stop timer"
}


/**
 * Production 32
 */
void initial_expression_list () :
{}
{
    initial_expression () ( "," initial_expression () )*
}


/**
 *  Production 33
 */
/** This production has two common prefix problems and and a left recursion problem and is modified */
/*
void initial_expression () :
{}
{
    < TRUE >
  | < FALSE >
  | < INTEGER_LITERAL >
  | < REAL_LITERAL >
  | < STRING_LITERAL >
  | id ()
  | type_name () "." op_name () [ "(" initial_expression_list () ")" ] \
  | "(" initial_expression () ")"
  | initial_expression () binary_op () initial_expression ()
  | unary_op () initial_expression ()
}
*/

void initial_expression () :
{}
{
    initial_expression_1 () initial_expression_tail ()
}

void initial_expression_1 () :
{}
{
    < TRUE >
  | < FALSE >
  | < STRING_LITERAL >
  | < INTEGER_LITERAL > initial_expression_suffix1 ()
  | id () initial_expression_suffix2 ()
  | "(" initial_expression () ")"
  | unary_op () initial_expression ()
}

void initial_expression_tail () :
{}
{
    binary_op () initial_expression () initial_expression_tail ()
  | empty_string ()
}

void initial_expression_suffix1 () :
{}
{
    empty_string ()
  | "." < INTEGER_LITERAL >
}

void initial_expression_suffix2 () :
{}
{
    empty_string ()
  | [ "[" type_name_suffix () "]" ] "." op_name () [ "(" initial_expression_list () ")" ]
}


/**
 * Production 34
 */
void binary_op () :
{}
```

63

```
{
   <AND> | <OR> | <XOR> | <GREATER_THAN> | <LESS_THAN>
   | <EQUALS> | <GREATER_OR_EQUAL_TO> | <LESS_OR_EQUAL_TO>
   | <DIVIDE_EQUALS> | <PLUS> | <MINUS> | <AMPERCENT>
   | <STAR> | <FACTOR> | <MOD> | <REM> | <STAR_STAR>
}


/**
  * Production 35
  */
void unary_op () :
{}
{
   <NOT> | <ABS> | <MINUS> | <PLUS>
}


/**
  *  Production 36
  */
void time () :
{}
{
   < INTEGER_LITERAL > unit ()
}


/**
  * Production 37
  */
void unit () :
{}
{
   <MICROSEC>
   | <MS>
   | <SEC>
   | <MIN>
   | <HOURS>
}


/**
  * Production 38
  */
void expression_list () :
{}
{
   expression () ( "," expression () )*
}


/**
  * Production 39
  */
/** This production has two common prefix problems and and a left recursion problem and is modified */
/*
void expression () :
{}
{
   <TRUE>
   | <FALSE>
   | time ()
   | < INTEGER_LITERAL >
   | < REAL_LITERAL >
   | < STRING_LITERAL >
   | id ()
   | type_name () "." op_name () [ "(" expression_list () ")" ]
   | "(" expression () ")"
   | expression () binary_op () expression ()
   | unary_op () expression ()
}
*/

void expression () :
{}
{
   expression_1 () expression_tail ()
}

void expression_1 () :
{}
{
   < TRUE >
   | < FALSE >
   | < STRING_LITERAL >
   | < INTEGER_LITERAL > expression_suffix1 ()
   | id () expression_suffix2 ()
   | "(" expression () ")"
   | unary_op () expression ()
}

void expression_tail () :
{}
{
   binary_op () expression () expression_tail ()
   | empty_string ()
}

void expression_suffix1 () :
```

```
{}
{
   empty_string ()
 | "." < INTEGER_LITERAL >
 | unit ()
}

void expression_suffix2 () :
{}
{
   empty_string ()
 | [ "[" type_name_suffix () "]" ] "." op_name () [ "(" expression_list () ")" ]
}

/**
  * Production 40
  */
void op_name () :
{}
{
   id ()
}

/**
  * Production 41
  */
void id () :
{}
{
   < IDENTIFIER >
}

/*
   Production 42
*/
/* This is a token, it is removed from the parser for efficiency
void real_literal () :
{}
{
   < INTEGER_LITERAL > "." < INTEGER_LITERAL >
}
*/

/**
  * Production 43
  */
void integer_literal () :
{}
{
   < INTEGER_LITERAL >
}

/*
   Production 44
*/
/* This is a token, it is removed from the parser for efficiency
void string_literal () :
{}
{
   < STRING_LITERAL >
}
*/

/*
   Production 45
*/
/* This is a token, it is removed from the parser for efficiency
void digit () :
{}
{
   < DIGIT >
}
*/

/*
   Production 46
*/
/* This is a token, it is removed from the parser for efficiency
void letter () :
{}
{
   < LETTER >
}
*/

/*
   Production 47
*/
/* This goes into < IDENTIFIER >, it is removed from the parser
void alpha_numeric () :
{}
{
   letter ()
 | digit ()
 | "_"
```

```
}
*/

/*
 * Production 48
 */
/* This production goes into < TEXT > */
/*
void text () :
{}
{
   < TEXT >
}
*/

/**
  * Production 49
  */
/** Represents the empty string, not a part of the PSDL grammar */
void empty_string () :
{}
{
   { return; }
}

/*
   Production 50
*/
/* This production is no more needed
void ch_ar () :
{}
{
   < CHAR : ~["}"] >
}
*/

/**
  * This production is not in the grammar
  * It is used to check output guards of a vertex
  */
void check_output_guards () :
{}
{
   ( < OUTPUT > id_list () < IF > expression () [reqmts_trace ()] )+
}

/**
  * This production is not in the grammar
  * It is used to check exception guards of a vertex
  */
void check_exception_guards () :
{}
{
   ( < EXCEPTION > id () [ < IF > expression () ] [reqmts_trace ()] )+
}

/**
  * This production is not in the grammar
  * It is used to check exception list of a vertex
  */
void check_exception_list () :
{}
{
   ( < EXCEPTIONS > id_list () )+
}

/**
  * This production is not in the grammar
  * It is used to check timer ops of a vertex
  */
void check_timer_ops () :
{}
{
   ( timer_op () id () [ < IF > expression () ] [reqmts_trace ()] )+
}

/**
  * This production is not in the grammar
  * It is used to check parent specs
  */
void check_parent_spec () :
{}
{
   < OPERATOR > op_name () operator_spec ()
}
```

```
/*
   Program : PsdlBuiler.jj
   Author  : Ilker Duranlioglu
   This grammar file is written in JavaCC version 0.8pre1.
*/

options {
   IGNORE_CASE = true;                /* PSDL is not case sensitive              */
   DEBUG_PARSER = true;               /* Set this flag to true to trace the parser calls */
}


PARSER_BEGIN (PsdlBuilder)

package caps.Builder;

import java.io.*;
import java.util.*;
import caps.Psdl.*;

public class PsdlBuilder {

   private static Vector dfcVector;

   private static Vector streamsVector;

   private static Vertex currentOp;

   private static Vertex currentChild;

   private static Edge currentEdge;

   private static Vector idList = new Vector ();

   private static int index;

   public static void main (String args[]) throws ParseException {
   }

   public static Vertex buildPrototype (File file)
   {
      BufferedReader reader = null;
      try {
         reader = new BufferedReader (new FileReader (file));
      } catch (FileNotFoundException e) {
         System.out.println (e);
      }
      dfcVector = new Vector ();
      streamsVector = new Vector ();
      idList = new Vector ();
      currentOp = null;
      currentEdge = null;
      currentChild = null;
      ReInit (reader);

      try {
         psdl ();
      } catch (ParseException ex) {
         System.out.println (ex);
         System.exit (0);
      }

      Vertex root = findRoot ();
      dfcVector = null;
      idList = null;
      currentOp = null;
      currentEdge = null;
      currentChild = null;
      return root;
   }

   public static String label;

   public static int id;

   public static Vertex findOperator (String name, boolean doubleSuffix)
   {
      DataFlowComponent d;
      Vertex found = null;
      extractLabel (name, doubleSuffix);
      for (Enumeration enum = dfcVector.elements (); enum.hasMoreElements ();) {
         d = (DataFlowComponent) enum.nextElement ();
         String str = "";
         if (doubleSuffix)
            str = new String (d.getLabel () + "_" + d.getId () + "_" + (d.getId () - 1));
         else
            str = new String (d.getLabel () + "_" + d.getId ());
         if ((d instanceof Vertex) && (str.equals (name)))
            found = (Vertex) d;
      }
      if (found == null) {
         if (doubleSuffix == false)
            found = new Vertex (0, 0, null, false);      // This is the root
         else
            found = new Vertex (0, 0, currentOp, false);    // This is a child vertex
```

```
            found.setLabel (label);
            found.setId (id);
            dfcVector.addElement (found);
        }
        else if (doubleSuffix && found.getParent () == null) {
            currentOp.add (found);
        }
        return found;
    }

    public static void extractLabel (String s, boolean doubleSuffix)
    {
        int index = s.lastIndexOf ("_");
        String temp = s.substring (index + 1, s.length ());
        int num = Integer.parseInt (temp);
        s = new String (s.substring (0, index));
        if (doubleSuffix == false) {   // If contains only one suffix
            label = s;
            id = num;
        }
        else {
            index = s.lastIndexOf ("_");
            temp = s.substring (index + 1, s.length ());
            num = Integer.parseInt (temp);
            s = new String (s.substring (0, index));
            label = s;
            id = num;
        }
    }

    public static Edge findEdge (String name)
    {
        DataFlowComponent d;
        Edge found = null;
        for (Enumeration enum = streamsVector.elements (); enum.hasMoreElements ();) {
            d = (DataFlowComponent) enum.nextElement ();
            if ((d instanceof Edge) && (d.getLabel ().equals (name)))
                found = (Edge) d;
        }
        return found;
    }

    public static Vertex findRoot ()
    {
        Vertex o = null;
        DataFlowComponent d;
        for (Enumeration enum = dfcVector.elements ();enum.hasMoreElements ();) {
            d = (DataFlowComponent) enum.nextElement ();
            if (d.getParent () == null)
                o = (Vertex) d;
        }
        return o;
    }

    public static String extractIdList (Vector v)
    {
        String str = "";
        Enumeration enum;
        if (v != null) {
            enum = v.elements ();
            if (enum.hasMoreElements ())
                str = new String ((String) enum.nextElement ());
            while (enum.hasMoreElements ()) {
                str = str.concat (", ").concat ((String) enum.nextElement ());
            }
        }
        return str;
    }

    public static Vertex findChild (String name)
    {
        DataFlowComponent d;
        Vertex found = null;
        extractLabel (name, true);           // DoubleSuffix
        for (Enumeration enum = currentOp.children (); enum.hasMoreElements ();) {
            d = (DataFlowComponent) enum.nextElement ();
            if ((d instanceof Vertex) && (d.getLabel ().equals (label)))
                found = (Vertex) d;
        }
        return found;
    }

    public static void setCurrentOp (Vertex v)
    {
        currentOp = v;
    }

    public static void setVertexProperty (Vertex v, String prop, String exp)
    {
        if (prop.equalsIgnoreCase ("x"))
            v.setX (Integer.parseInt (exp));
        else if (prop.equalsIgnoreCase ("y"))
            v.setY (Integer.parseInt (exp));
        else if (prop.equalsIgnoreCase ("radius"))
            v.setWidth (Integer.parseInt (exp) * 2);
```

```
          else if (prop.equalsIgnoreCase ("color"))
              v.setColor (Integer.parseInt (exp));
          else if (prop.equalsIgnoreCase ("label_font"))
              v.setLabelFontIndex (Integer.parseInt (exp));
          else if (prop.equalsIgnoreCase ("label_x_offset"))
              v.setLabelXOffset (Integer.parseInt (exp));
          else if (prop.equalsIgnoreCase ("label_y_offset"))
              v.setLabelYOffset (Integer.parseInt (exp));
          else if (prop.equalsIgnoreCase ("met_font"))
              v.setMetFontIndex (Integer.parseInt (exp));
          else if (prop.equalsIgnoreCase ("met_unit")) {
              if (v.getMet () != null)
                  v.getMet ().setTimeUnits (exp);
          }
          else if (prop.equalsIgnoreCase ("met_x_offset"))
              v.setMetXOffset (Integer.parseInt (exp));
          else if (prop.equalsIgnoreCase ("met_y_offset"))
              v.setMetYOffset (Integer.parseInt (exp));
          else if (prop.equalsIgnoreCase ("is_terminator")) {
              if (exp.equalsIgnoreCase ("true"))
                  v.setTerminator (true);
          }
      }

      public static void setEdgeProperty (Edge e, String prop, String exp)
      {
          if (prop.equalsIgnoreCase ("id"))
              e.setId (Integer.parseInt (exp));
          else if (prop.equalsIgnoreCase ("label_font"))
              e.setLabelFontIndex (Integer.parseInt (exp));
          else if (prop.equalsIgnoreCase ("label_x_offset"))
              e.setLabelXOffset (Integer.parseInt (exp));
          else if (prop.equalsIgnoreCase ("label_y_offset"))
              e.setLabelYOffset (Integer.parseInt (exp));
          else if (prop.equalsIgnoreCase ("latency_font"))
              e.setMetFontIndex (Integer.parseInt (exp));
          else if (prop.equalsIgnoreCase ("latency_unit")) {
              if (e.getMet () != null)
                  e.getMet ().setTimeUnits (exp);
          }
          else if (prop.equalsIgnoreCase ("latency_x_offset"))
              e.setMetXOffset (Integer.parseInt (exp));
          else if (prop.equalsIgnoreCase ("latency_y_offset"))
              e.setMetYOffset (Integer.parseInt (exp));
          else if (prop.equalsIgnoreCase ("spline"))
              e.setInitialControlPoints (exp);
      }

} // End of the class PsdlBuilder

PARSER_END (PsdlBuilder)


/* White Space */
SKIP :
{
    " "
|   "\r"
|   "\t"
|   "\n"
}

/* Reserved Words */
TOKEN :
{
    < IF : "if" >
|   < MS : "ms" >
|   < SEC : "sec" >
|   < END : "end" >
|   < MIN : "min" >
|   < TYPE : "type" >
|   < EDGE : "edge" >
|   < TRUE : "true" >
|   < FALSE : "false" >
|   < GRAPH : "graph" >
|   < TIMER : "timer" >
|   < HOURS : "hours" >
|   < INPUT : "input" >
|   < PERIOD : "period" >
|   < STATES : "states" >
|   < AXIOMS : "axioms" >
|   < OUTPUT : "output" >
|   < VERTEX : "vertex" >
|   < GENERIC : "generic" >
|   < MICROSEC : "microsec" >
|   < OPERATOR : "operator" >
|   < KEYWORDS : "keywords" >
|   < PROPERTY : "property" >
|   < TRIGGERED : "triggered" >
|   < EXCEPTION : "exception" >
|   < INITIALLY : "initially" >
|   < EXCEPTIONS : "exceptions" >
|   < DESCRIPTION : "description" >
|   < SPECIFICATION : "specification" >
|   < IMPLEMENTATION : "implementation" >
```

69

```
}

/* Operators */
TOKEN :
{
/* Binary Operators */
    < OR : "or" >
  | < AND : "and" >
  | < MOD : "mod" >
  | < REM : "rem" >
  | < XOR : "xor" >
  | < GREATER_THAN : ">" >
  | < LESS_THAN : "<" >
  | < EQUALS : "=" >
  | < GREATER_OR_EQUAL_TO : ">=" >
  | < LESS_OR_EQUAL_TO : "<=" >
  | < DIVIDE_EQUALS : "/=" >
  | < PLUS : "+" >
  | < MINUS : "-" >
  | < AMPERCENT : "&" >
  | < STAR : "*" >
  | < FACTOR : "/" >
  | < STAR_STAR : "**" >
/* Unary Operators */
  | < ABS : "abs" >
  | < NOT : "not" >
}


/* String real, and integer literals */
TOKEN :
{
    < TEXT : "{" ( < CHAR_TEXT > )* "}" >
  | < #CHAR_TEXT : ~["}"] >

  | < STRING_LITERAL : "\"" ( < CHAR_LIT > )* "\"" >
  | < #CHAR_LIT : ~["}", "\""] >

  | < REAL_LITERAL : < INTEGER_LITERAL > "." < INTEGER_LITERAL > >

  | < INTEGER_LITERAL : < INT_DIGIT > ( < INT_DIGIT > )* >
  | < #INT_DIGIT : ["0" - "9"] >
}

/* Identifiers */
TOKEN :
{
    < IDENTIFIER : < ID_LETTER > ( < ID_LETTER > | < ID_DIGIT > | "_" )* >
  | < #ID_LETTER : ["a" - "z"] | ["A" - "Z"] >
  | < #ID_DIGIT : ["0" - "9"] >
}

/* Digits and letters */
TOKEN :
{
    < DIGIT : ["0" - "9"] >
  | < LETTER : ["a" - "z"] | ["A" - "Z"]>
}


/**
 * Production 1
 */
void psdl () :
{}
{
    ( component () )*
}

/**
 *  Production 2
 */
void component () :
{}
{
    data_type ()
  | operator ()
}

/**
 * Production 3
 */
void data_type () :
{}
{
    <TYPE> id () type_spec () type_impl ()
}

/**
 * Production 4
 */
/* <functionality> is directly included in this production */
void type_spec () :
{}
{
    <SPECIFICATION> [ <GENERIC> type_decl (false) ] [ type_decl (false) ]
```

```
    ( <OPERATOR> op_name () operator_spec () )*
    [ keywords () ] [informal_desc () ] [ formal_desc () ] <END>
}


/**
 * Production 5
 */
void operator () :
{
    String name;
}
{
    <OPERATOR> name = op_name ()
    { currentOp = findOperator (name, false); }
    operator_spec () operator_impl ()
    {
        currentOp = null;
    }
}


/**
 * Production 6
 */
/* <functionality> is directly included in this production */
void operator_spec () :
{
    String desc;
    Vector list;
}
{
    <SPECIFICATION> ( inter_face () )*
    [ list = keywords () { currentOp.setKeywordList (list); } ]
    [ desc = informal_desc () { currentOp.setInformalDesc (desc); } ]
    [ desc = formal_desc () { currentOp.setFormalDesc (desc); } ] <END>
}


/**
 * Production 7
 */
void inter_face () :
()
{
    attribute () /* reqmts_trace is under attribute */
}


/**
 * Production 8
 */
void attribute () :
{
    String initial;
    Token tok;
    Vector list;
    PSDLTime met;
    Vector reqmts = null;
    String str;
}
{
    tok = <GENERIC> type_decl (false)
    { str = currentOp.getGenericList ();
      if (str != "")
        str = str.concat ("\n");
      str = str.concat (tok.toString () + " " + extractIdList (idList));
    }
    [ list = reqmts_trace () { str = str.concat ("\n   REQUIRED BY " + extractIdList (list)); } ]
    { currentOp.setGenericList (str); }
  | <INPUT> type_decl (false) [ reqmts = reqmts_trace () ]
    { ((Vector) currentOp.getSpecReqmts ().elementAt (0)).addElement (extractIdList (reqmts)); }
  | <OUTPUT> type_decl (false) [ reqmts = reqmts_trace () ]
    { ((Vector) currentOp.getSpecReqmts ().elementAt (1)).addElement (extractIdList (reqmts)); }
  | <STATES> type_decl (true) <INITIALLY> initial = initial_expression_list ()
    {
        currentEdge.setStateStream (true);
        currentEdge.setInitialValue (initial);
    }
    [ reqmts = reqmts_trace () ]
    { ((Vector) currentOp.getSpecReqmts ().elementAt (2)).addElement (extractIdList (reqmts)); }
  | tok = <EXCEPTIONS> list = id_list ()
    { str = currentOp.getExceptionList ();
      if (str != "")
        str = str.concat ("\n");
      str = str.concat (tok.toString () + " " + extractIdList (list));
    }
    [ list = reqmts_trace () { str = str.concat ("\n   REQUIRED BY " + extractIdList (list)); } ]
    { currentOp.setExceptionList (str); }
  | "maximum execution time" met = time () { currentOp.setMet (met); }
    [ list = reqmts_trace () { currentOp.setMetReqmts (list); } ]
}


/**
 * Production 9
 */
void type_decl (boolean buildEdge) :
{
Vector idList;
```

71

```
   String type = "";
   }
   {
      idList = id_list () ":" type = type_name ()
      {
         currentEdge = findEdge ((String) idList.elementAt (0));
         if (buildEdge && (currentEdge == null)) {
            currentEdge = new Edge (0, 0, currentOp);
            currentEdge.setLabel ((String) idList.elementAt (0));
            streamsVector.addElement (currentEdge);
         }
         if (buildEdge)
            currentEdge.setStreamType (type);
      }
      ( ","  idList = id_list () ":" type = type_name ()
      {
         if (buildEdge) {
            currentEdge = findEdge ((String) idList.elementAt (0));
            for (Enumeration enum = streamsVector.elements (); enum.hasMoreElements ();) {
               Edge e = (Edge) enum.nextElement ();
               if (currentEdge.getLabel ().equals (e.getLabel ()))
                  e.setStreamType (type);
            }
         }
      }
      )*
   }

/**
  * Production 10
  */
/* This production is mofidied to remove common prefix id () */
/*
String type_name () :
{
   String name = "";
}
{
   name = id ()   { return name;}
 | id () "[" type_decl (false) "]"
}
*/

String type_name () :
{
   String name;
}
{
   name = id () type_name_suffix ()
   { return name;}
}

/** This production is to remove the common prefix id () */
void type_name_suffix () :
{}
{
   "[" type_decl (false) "]"
 | empty_string ()
}

/**
  * Production 11
  */
Vector id_list () :
{
   idList = new Vector ();
   String name;
}
{
   name = id () { idList.addElement (name); }
   ( "," name = id () { idList.addElement (name); } )*
   {return idList; }
}

/**
  * Production 12
  */
Vector reqmts_trace () :
{
   Vector list;
}
{
   "required by" list = id_list ()
   { return list; }
}

/**
  * Production 13
  */
/* This production is included directly in other productions,
   because it caused empty string
void functionality () :
{}
{
   [ keywords () ] [informal_desc () ] [ formal_desc () ]
```

```
}
*/

/**
 * Production 14
 */
Vector keywords () :
{
    Vector list;
}
{
    <KEYWORDS> list = id_list ()
    { return list; }
}

/**
 * Production 15
 */
String informal_desc () :
{
    Token tok;
    Token text;
}
{
    tok = <DESCRIPTION> text = < TEXT >
    { return new String (tok.toString () + " " + text.toString ()); }
}

/**
 * Production 16
 */
String formal_desc () :
{
    Token tok;
    Token text;
}
{
    tok = <AXIOMS> text = < TEXT >
    { return new String (tok.toString () + " " + text.toString ()); }
}

/**
 * Production 17
 */
/* This production is causing a common prefix problem and is modified
void type_impl () :
()
{
    <IMPLEMENTATION> id () id () <END>
  | <IMPLEMENTATION> type_name ()
    ( <OPERATOR> op_name () operator_impl () )* <END>
}
*/

void type_impl () :
()
{
    <IMPLEMENTATION> id () type_impl_suffix ()
}

/** This production is to remove the common prefix "implementation" */
void type_impl_suffix () :
()
{
    id () <END>
  | [ "[" type_name_suffix () "]" ] ( <OPERATOR> op_name () operator_impl () )* <END>
}

/**
 * Production 18
 */
/* This production causes a common prefix problem and hence is modified
void operator_impl () :
()
{
    <IMPLEMENTATION> id () id () <END>
  | <IMPLEMENTATION> psdl_impl () <END>
}
*/

void operator_impl () :
()
{
    <IMPLEMENTATION> operator_impl_suffix ()
}

/** This production is to remove common prefix "implementation" */
void operator_impl_suffix () :
{
    String language;
}
{
    language = id () id () <END>
    { currentOp.setImpLanguage (language); }
  | psdl_impl () <END>
```

```
}

/**
 * Production 19
 */
void psdl_impl () :
{
    String desc;
}
{
    data_flow_diagram () [ streams () ][ timers () ]
    [ control_constraints () ]
    [ desc = informal_desc () { currentOp.setGraphDesc (desc); } ]
}

/**
 * Production 20
 */
void data_flow_diagram () :
{}
{
    <GRAPH> ( vertex () )* ( edge () )*
}

/**
 * Production 21
 */
void vertex () :
{
    String name;
    PSDLTime met;
}
{
    <VERTEX> name = op_id ()
        { currentChild = findOperator (name, true); }
    [ ":" met = time () { currentChild.setMet (met); } ] ( property (currentChild) )*
    { currentChild = null; }
}

/**
 * Production 22
 */
void edge () :
{
    String name;
    PSDLTime latency;
    Vertex src;
    Vertex dest;
}
{
    <EDGE> name = id ()
        { if ((currentEdge = findEdge (name)) == null) {
            currentEdge = new Edge (0, 0, currentOp);
            currentEdge.setLabel (name);
            streamsVector.addElement (currentEdge);
          }
          else {
            if (currentEdge.getSource () == null) {
                streamsVector.removeElement (currentEdge);
                currentEdge.removeFromParent ();
            }
            Edge e = new Edge (0, 0, currentOp);
            e.setLabel (name);
            e.setStreamType (currentEdge.getStreamType ());
            e.setStateStream (currentEdge.isStateStream ());
            e.setInitialValue (currentEdge.getInitialValue ());
            streamsVector.addElement (e);
            currentEdge = e;
          }
        }
    [ ":" latency = time () { currentEdge.setMet (latency); } ]
    name = op_id () { if (name.equals ("EXTERNAL")) {
                        External ex = new External (0, 0, currentOp);
                        ex.addOutEdge (currentEdge);
                        currentEdge.setSource (ex);
                    }
                    else {
                        src = findOperator (name, true);
                        currentEdge.setSource (src);
                        src.addOutEdge (currentEdge);
                    }
                }
    "->"
    name = op_id () { if (name.equals ("EXTERNAL")) {
                        External ex = new External (0, 0, currentOp);
                        ex.addInEdge (currentEdge);
                        currentEdge.setDestination (ex);
                    }
                    else {
                        dest = findOperator (name, true);
                        currentEdge.setDestination (dest);
                        dest.addInEdge (currentEdge);
                    }
                }
    ( property (currentEdge) )*
```

74

```
}

/**
  * Production 23
  */
void property (DataFlowComponent dfc) :
{
    String prop;
    String exp;
}
{
    <PROPERTY> prop = id () "=" exp = expression ()
    { if (dfc instanceof Vertex)
        setVertexProperty ((Vertex) dfc, prop, exp);
      else
        setEdgeProperty ((Edge) dfc, prop, exp);
    }
}

/**
  * Production 24
  */
/* This production has common prefix problem and is modified
void op_id () :
{}
{
    [ id () "." ] op_name () [ "(" [ id_list () ] "|" [ id_list () ] ")" ]
}
*/

String op_id () :
{
    String name;
}
{
    name = op_name () [ "." id () ] [ "(" [ id_list () ] "|" [ id_list () ] ")" ]
    { return name; }
}

/**
  * Production 25
  */
void streams () :
{}
{
    "data stream" type_decl (true)
    { streamsVector.removeAllElements (); }
}

/**
  * Production 26
  */
void timers () :
{
    Vector list;
}
{
    <TIMER> list = id_list () { currentOp.setTimerList (list); }
}

/**
  * Production 27
  */
void control_constraints () :
{}
{
    "control constraints" constraint () ( constraint () )*
}

/**
  * Production 28
  */
void constraint () :
{
    Vector list;
    String str;
    PSDLTime t;
}
{
    <OPERATOR> str = op_id ()
    { currentChild = findChild (str); }
    [ <TRIGGERED> [ list = trigger () { currentChild.setTriggerType (index);
                                        currentChild.setTriggerStreamsList (list); } ]
      [ <IF> str = expression () { currentChild.setIfCondition (str); }]
      [ list = reqmts_trace () { currentChild.setTriggerReqmts (list); } ] ]
    [ <PERIOD> t = time () { currentChild.setTimingType (Vertex.PERIODIC);
                            currentChild.setPeriod (t); }
      [ list = reqmts_trace () { currentChild.setPeriodReqmts (list); } ] ]
    [ "finish within" t = time () { currentChild.setFinishWithin (t); }
      [ list = reqmts_trace () { currentChild.setFinishWithinReqmts (list); } ] ]
    [ "minimum calling period" t = time () { currentChild.setTimingType (Vertex.SPORADIC);
                                            currentChild.setMcp (t); }
      [ list = reqmts_trace () { currentChild.setMcpReqmts (list); } ] ]
    [ "maximum response time" t = time () { currentChild.setMrt (t); }
      [ list = reqmts_trace () { currentChild.setMrtReqmts (list); }] ]
```

```
    ( constraint_options () )*
}

/**
 * Production 29
 */
void constraint_options () :
{
    Token tok;
    Vector list;
    String expr = "";
    String str = "";
}
{
    tok = <OUTPUT> list = id_list () { str = new String (tok.toString () + " " + extractIdList (list)); }
    tok = <IF> expr = expression () { str = new String (str + " " + tok.toString () + " " + expr); }
    [ list = reqmts_trace () { str = new String (str + "\n    " + "REQUIRED BY " + extractIdList (list)); }]
    { currentChild.setOutputGuardList (str); }
  | tok = <EXCEPTION> expr = id () { str = new String (tok.toString () + " " + expr); }
    [ tok = <IF> expr = expression () { str = new String (str + " " + tok.toString () + " " + expr); } ]
    [ list = reqmts_trace () { str = new String (str + "\n    " + "REQUIRED BY " + extractIdList (list)); } ]
    { currentChild.setExceptionGuardList (str); }
  | str = timer_op () expr = id () { str = new String (str + " " + expr); }
    [ tok = <IF> expr = expression () { str = new String (str + " " + tok.toString () + " " + expr); } ]
    [ list = reqmts_trace () { str = new String (str + "\n    " + "REQUIRED BY " + extractIdList (list)); } ]
    { currentChild.setTimerOpList (str); }
}

/**
 * Production 30
 */
Vector trigger () :
{
    Vector list;
}
{
    "by all" list = id_list ()
    { index = 2;
      return list; }
  | "by some" list = id_list ()
    { index = 1;
      return list; }
}

/**
 * Production 31
 */
String timer_op () :
{}
{
    "reset timer" { return new String ("RESET TIMER"); }
  | "start timer" { return new String ("START TIMER"); }
  | "stop timer"  { return new String ("STOP TIMER"); }
}

/**
 * Production 32
 */
String initial_expression_list () :
{
    String list = "";
    String expr = "";
}
{
    list = initial_expression () ( "," expr = initial_expression ()
              { list = list.concat (", ").concat (expr); }
                                  )*
    { return list; }
}

/**
 *  Production 33
 */
/** This production has two common prefix problems and and a left recursion problem and is modified */
/*
void initial_expression () :
{}
{
    < TRUE >
  | < FALSE >
  | < INTEGER_LITERAL >
  | < REAL_LITERAL >
  | < STRING_LITERAL >
  | id ()
  | type_name () "." op_name () [ "(" initial_expression_list () ")" ] \
  | "(" initial_expression () ")"
  | initial_expression () binary_op () initial_expression ()
  | unary_op () initial_expression ()
}
*/

String initial_expression () :
{
    String str = "";
    String tail = "";
```

76

```
}
{
    str = initial_expression_1 () tail = initial_expression_tail ()
    { return new String (str + tail); }
}

String initial_expression_1 () :
{
    Token tok;
    String str;
    String suffix = "";
}
{
    tok = < TRUE > { return tok.toString (); }
  | tok = < FALSE > { return tok.toString (); }
  | tok = < STRING_LITERAL > { return tok.toString (); }
  | tok = < INTEGER_LITERAL > suffix = initial_expression_suffix1 ()
    { return new String (tok.toString () + suffix); }
  | str = id () suffix = initial_expression_suffix2 ()
    { return new String (str + suffix); }
  | "(" str = initial_expression () ")"
    { return new String ("(" + str + ")"); }
  | str = unary_op () suffix = initial_expression ()
    { return new String (str + suffix); }
}

String initial_expression_tail () :
{
    String str;
    String suffix1;
    String suffix2;
}
{
    str = binary_op () suffix1 = initial_expression () suffix2 = initial_expression_tail ()
    { return new String (str + suffix1 + suffix2); }
  | empty_string ()
    { return ""; }
}

String initial_expression_suffix1 () :
{
    Token tok;
}
{
    empty_string ()
    { return ""; }
  | "." tok = < INTEGER_LITERAL >
    { return new String ("." + tok.toString ()); }
}

String initial_expression_suffix2 () :
{
    String str = "";
    String s = "";
}
{
    empty_string ()
    { return ""; }
  | [ "[" str = type_name_suffix () "]" { str = new String ("[" + str + "]"); } ] "." s = op_name ()
    { str = str.concat (".").concat (s); }
    [ "(" s = initial_expression_list () ")" { str = new String (str + "(" + s + ")"); }]
    { return str; }

}

/**
 * Production 34
 */
String binary_op () :
{
    Token tok;
}
{
    tok = <AND> { return tok.toString (); }
  | tok = <OR> { return tok.toString (); }
  | tok = <XOR> { return tok.toString (); }
  | tok = <GREATER_THAN> { return tok.toString (); }
  | tok = <LESS_THAN> { return tok.toString (); }
  | tok = <EQUALS> { return tok.toString (); }
  | tok = <GREATER_OR_EQUAL_TO> { return tok.toString (); }
  | tok = <LESS_OR_EQUAL_TO> { return tok.toString (); }
  | tok = <DIVIDE_EQUALS> { return tok.toString (); }
  | tok = <PLUS> { return tok.toString (); }
  | tok = <MINUS> { return tok.toString (); }
  | tok = <AMPERCENT> { return tok.toString (); }
  | tok = <STAR> { return tok.toString (); }
  | tok = <FACTOR> { return tok.toString (); }
  | tok = <MOD> { return tok.toString (); }
  | tok = <REM> { return tok.toString (); }
  | tok = <STAR_STAR> { return tok.toString (); }
}

/**
 * Production 35
 */
```

```
String unary_op () :
{
    Token tok;
}
{
    tok = <NOT> { return tok.toString (); }
  | tok = <ABS> { return tok.toString (); }
  | tok = <MINUS> { return tok.toString (); }
  | tok = <PLUS> { return tok.toString (); }
}

/**
  *  Production 36
  */
PSDLTime time () :
{
    PSDLTime t = new PSDLTime ();
    String str = "";
    Token tok;
}
{
    tok = < INTEGER_LITERAL >  { t.setTimeValue (Integer.parseInt (tok.toString ())); }
    str = unit () { t.setTimeUnits (str); }
    { return t; }
}

/**
  * Production 37
  */
String unit () :
{
    Token tok;
}
{
    tok = <MICROSEC> { return tok.toString (); }
  | tok = <MS> { return tok.toString (); }
  | tok = <SEC> { return tok.toString (); }
  | tok = <MIN> { return tok.toString (); }
  | tok = <HOURS> { return tok.toString (); }
}

/**
  * Production 38
  */
String expression_list () :
{
    String expList = "";
    String str = "";
}
{
    expList = expression () ( "," str = expression ()
            { expList = expList.concat (", ").concat (str); }
                                              )*
    {
        return expList;
    }
}

/**
  * Production 39
  */
/** This production has two common prefix problems and and a left recursion problem and is modified */
/*
void expression () :
()
{
    <TRUE>
  | <FALSE>
  | time ()
  | < INTEGER_LITERAL >
  | < REAL_LITERAL >
  | < STRING_LITERAL >
  | id ()
  | type_name () "." op_name () [ "(" expression_list () ")" ]
  | "(" expression () ")"
  | expression () binary_op () expression ()
  | unary_op () expression ()
}
*/

String expression () :
{
    String exp = "";
    String expTail = "";
}
{
    exp = expression_1 () expTail = expression_tail ()
    {
        exp = exp.concat (expTail);
        return exp;
    }
}

String expression_1 () :
{
```

```
    Token tok;
    String str = "";
    String suffix = "";
}
{
    tok = < TRUE > { return tok.toString (); }
  | tok = < FALSE > { return tok.toString (); }
  | tok = < STRING_LITERAL > { return tok.toString (); }
  | tok = < INTEGER_LITERAL > suffix = expression_suffix1 ()
    { return new String (tok.toString () + suffix); }
  | str = id () suffix = expression_suffix2 ()
    { return new String (str + suffix); }
  | "(" str = expression () ")"
    { /*return new String ("(" + str + ")");*/
        return new String (str); /* To accept -(15) */}
  | str = unary_op () suffix = expression ()
    { return new String (str + suffix); }
}

String expression_tail () :
{
    String str = "";
    String suffix1 = "";
    String suffix2 = "";
}
{
    str = binary_op () suffix1 = expression () suffix2 = expression_tail ()
    { return new String (str + suffix1 + suffix2); }
  | empty_string ()
    { return ""; }
}

String expression_suffix1 () :
{
    Token tok;
    String unit = "";
}
{
    empty_string ()
    { return ""; }
  | "." tok = < INTEGER_LITERAL >
    { return new String ("." + tok.toString ()); }
  | unit = unit ()
    { return unit; }
}

String expression_suffix2 () :
{
    String str = "";
    String s = "";
}
{
    empty_string ()
    { return ""; }
  | [ "[" str = type_name_suffix () "]" { str = new String ("[" + str + "]"); } ] "." s = op_name ()
    { str = str.concat (".").concat (s); }
    [ "(" s = expression_list () ")" { str = new String (str + "(" + s + ")"); }]
    { return str; }
}

/**
  * Production 40
  */
String op_name () :
{
    String name;
}
{
    name = id ()
    {
        return name;
    }
}

/**
  * Production 41
  */
String id () :
{
    Token tok;
}
{
    tok = < IDENTIFIER >
    {
        return tok.toString ();
    }
}

/*
  Production 42
*/
/* This is a token, it is removed from the parser for efficiency
void real_literal () :
{}
{
```

```
        < INTEGER_LITERAL > "." < INTEGER_LITERAL >
}
*/


/**
  * Production 43
  */
String integer_literal () :
{
   Token intLiteral;
}
{
   intLiteral = < INTEGER_LITERAL >
   {
      return intLiteral.toString ();
   }
}


/*
    Production 44
*/
/* This is a token, it is removed from the parser for efficiency
void string_literal () :
{}
{
   < STRING_LITERAL >
}
*/


/*
    Production 45
*/
/* This is a token, it is removed from the parser for efficiency
void digit () :
{}
{
   < DIGIT >
}
*/


/*
    Production 46
*/
/* This is a token, it is removed from the parser for efficiency
void letter () :
{}
{
   < LETTER >
}
*/


/*
    Production 47
*/
/* This goes into < IDENTIFIER >, it is removed from the parser
void alpha_numeric () :
{}
{
   letter ()
 | digit ()
 | "_"
}
*/


/*
  * Production 48
  */
/* This production goes into < TEXT > */
/*
String text () :
{
   Token text;
}
{
   text = < TEXT >
   {
      return text.toString ();
   }
}
*/


/**
  * Production 49
  */
/** Represents the empty string, not a part of the PSDL grammar */
void empty_string () :
{}
{
   {
      return;
   }
}


/*
    Production 50
```

```
*/
/* This production is no more needed
void ch_ar () :
{}
{
    < CHAR : ~["}"] >
}
*/


/**
 * This production is not in the grammar
 * It is used to check output guards of a vertex
 */
void check_output_guards () :
{}
{
    ( < OUTPUT > id_list () < IF > expression () [reqmts_trace ()] )+
}


/**
 * This production is not in the grammar
 * It is used to check exception guards of a vertex
 */
void check_exception_guards () :
{}
{
    ( < EXCEPTION > id () [ < IF > expression () ] [reqmts_trace ()] )+
}


/**
 * This production is not in the grammar
 * It is used to check exception list of a vertex
 */
void check_exception_list () :
{}
{
    ( < EXCEPTIONS > id_list () )+
}


/**
 * This production is not in the grammar
 * It is used to check timer ops of a vertex
 */
void check_timer_ops () :
{}
{
    ( timer_op () id () [ < IF > expression () ] [reqmts_trace ()] )+
}
```

APPENDIX C. DOCUMENTATION OF THE SOURCE CODE

83

Package caps

Class Index

- <u>Caps</u>
- <u>EditorDriver</u>

Package Index

- package **caps**
- package **caps.CAPSMain**
- package **caps.Display**
- package **caps.GraphEditor**
- package **caps.Psdl**

84

Package caps.CAPSMain

Class Index

- CAPSMainMenuBar
- CAPSMainWindow
- DatabasesMenu
- EditMenu
- ExecSupportMenu
- ExitCAPSMain
- HelpMenu
- PrototypeMenu

Package caps.Display

Class Index

- DisplayComponent
- DisplayExternal
- DisplayVertex
- EdgePath

85

Package caps.GraphEditor

Class Index

- ColorConstants
- DrawPanel
- EdgeProperties
- Editor
- EditorMenuBar
- ExitEditor
- FontConstants
- GE_EditMenu
- GE_FileMenu
- GE_HelpMenu
- GE_PSDLMenu
- GE_ViewMenu
- IdListEditor
- Popup
- PrintJob
- StatusBar
- TextEditor
- ToolBar
- TreePanel
- TreePanelRenderer
- VertexProperties

86

Class caps.Caps

```
java.lang.Object
   |
   +----caps.Caps
```

public class **Caps**

extends java.lang.Object

The driver program for CAPS.

Constructor Index

• Caps()

Method Index

• main(String[])

The constructor for this class.

Package caps.Psdl

Class Index

- DataFlowComponent
- DataTypes
- Edge
- External
- PSDLTime
- Vertex

Class caps.EditorDriver

```
java.lang.Object
   |
   +----caps.EditorDriver
```

public class **EditorDriver**

extends java.lang.Object

The driver class for the PSDL Editor. This class is intended to execute the Editor in a stand alone way for debugging purposes.

Constructor Index

● EditorDriver()

Method Index

● main(String[])

The main method for this class

Constructors

○ Caps

public **Caps**()

Methods

● **main**

public static void **main**(java.lang.String args[])

The constructor for this class.

Parameters:

args[] - The command line parameters. (No command line parameter is necessary for this program.)

Class caps.CAPSMain.CAPSMainMenuBar

```
java.lang.Object
   |
   +----java.awt.Component
        |
        +----java.awt.Container
             |
             +----javax.swing.JComponent
                  |
                  +----javax.swing.JMenuBar
                       |
                       +----
```

caps.CAPSMain.CAPSMainMenuBar

public class **CAPSMainMenuBar**

extends javax.swing.JMenuBar

The menubar of the main CAPS window.

Constructor Index

CAPSMainMenuBar(CAPSMainWindow)

The constructor for this class.

Constructors

EditorDriver
public **EditorDriver**()

Methods

●**main**
public static void **main**(java.lang.String args[])

The main method for this class

Parameters:

args - The command line arguments for the driver

Class caps.CAPSMain.CAPSMainWindow

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Frame
                                   |
                                   +----javax.swing.JFrame
                                           |
                                           +----
```

caps.CAPSMain.CAPSMainWindow

public class **CAPSMainWindow**

extends javax.swing.JFrame

The main CAPS window.

Variable Index

● openPrototypes

The Vector that holds references to the open prototypes

Constructors

○ **CAPSMainMenuBar**

public **CAPSMainMenuBar**(CAPSMainWindow owner)

The constructor for this class.

Parameters:

owner - The parent class which has declared this menubar.

90

- prototype

  The File that contains the PSDL prototype.

Constructor Index

- CAPSMainWindow()

  The constructor for this class.

Method Index

- editPrototype()

  Opens the graphics editor to edit a prototype.

- getOpenPrototypes()

  Returns the vector that holds the open prototype files.

- initialize()

  Initializes the CAPS main window.

- isOpenPrototypeSaved()

  Checks if the status of any of the open prototypes is 'saveRequired'.

- isPrototypeChanged()

  Checks whether or not the current prototype file is already used by a PSDL Editor.

- removeEditor(Editor)

  Removes one element from the openPrototypes vector.

- setPrototype(File)

  Sets the prototype file to the argument.

Variables

- **prototype**

  `private java.io.File` **prototype**

  The File that contains the PSDL prototype.

- **openPrototypes**

  `private static java.util.Vector` **openPrototypes**

  The Vector that holds references to the open prototypes

Constructors

- CAPSMainWindow

  `public` **CAPSMainWindow**`()`

  The constructor for this class.

Methods

- **initialize**

  `public void` **initialize**`()`

  Initializes the CAPS main window.

- **setPrototype**

  `public void` **setPrototype**`(java.io.File f)`

  Sets the prototype file to the argument.

Parameters:

f - The File that contains the PSDL prototype.

●getOpenPrototypes

`public java.util.Vector getOpenPrototypes()`

Returns the vector that holds the open prototype files.

Returns:

the vector that contains the open prototype files.

●editPrototype

`public void editPrototype()`

Opens the graphics editor to edit a prototype. ●

isPrototypeChanged

`public boolean isPrototypeChanged()`

Checks whether or not the current prototype file is already used by a PSDL Editor.

Returns:

true if one of the open prototypes is the same as the current prototype file.

●removeEditor

`public static void removeEditor(Editor e)`

Removes one element from the openPrototypes vector.

Parameters:

e - the editor that is going to be removed from the vector.

●isOpenPrototypeSaved

`public boolean isOpenPrototypeSaved()`

Checks if the status of any of the open prototypes is 'saveRequired'. Prompts the user to save the prototype.

Returns:

true if none of the prototypes need saving.

Class caps.CAPSMain.DatabasesMenu

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----javax.swing.JComponent
                           |
                           +----javax.swing.AbstractButton
                                   |
                                   +----javax.swing.JMenuItem
                                           |
                                           +----javax.swing.JMenu
                                                   |
                                                   +----caps.CAPSMain.DatabasesMenu
```

public class **DatabasesMenu**

extends javax.swing.JMenu

implements java.awt.event.ActionListener

This class holds the 'Databases' menu items.

Variable Index

● **designDBMenuItem**

Initiates the 'Design Database' event

● **swBaseMenuItem**

Initiates the 'Software Base' event

Constructor Index

● **DatabasesMenu()**

Constructor for this class.

Method Index

● **actionPerformed**(ActionEvent)

Action event handler for the menu events.

Variables
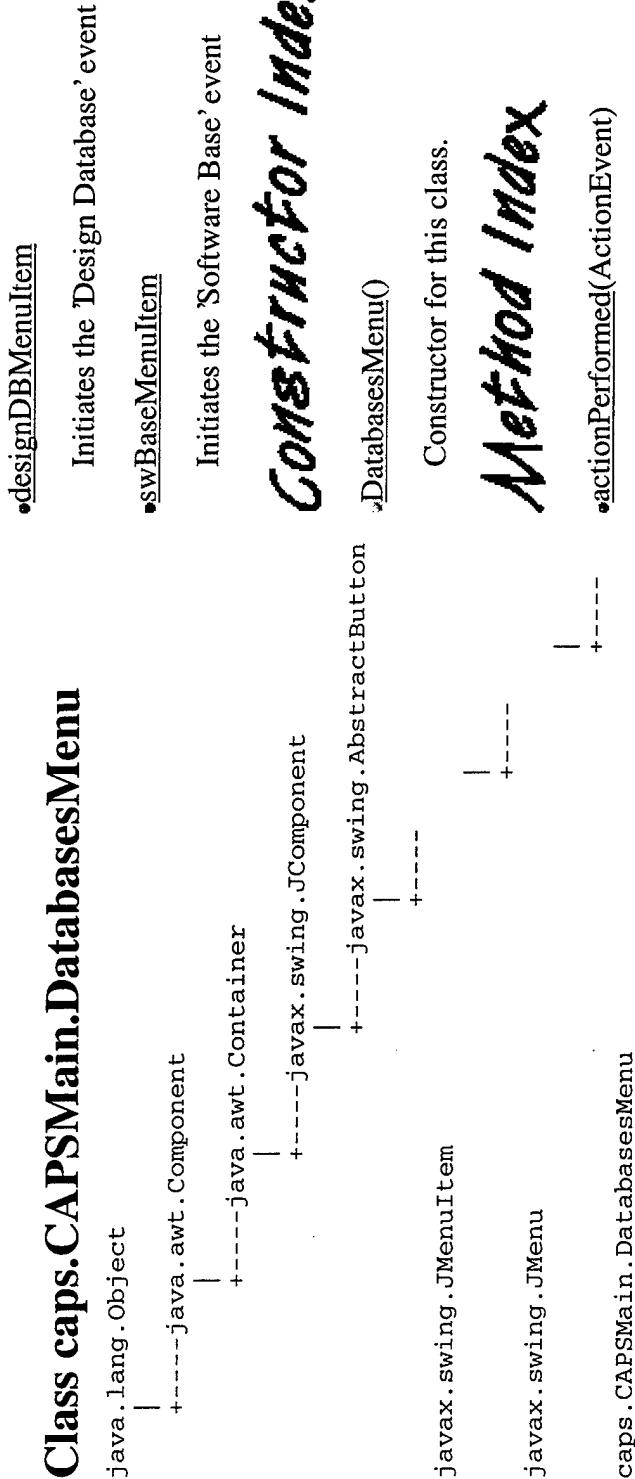
● **designDBMenuItem**

private javax.swing.JMenuItem **designDBMenuItem**

Initiates the 'Design Database' event

● **swBaseMenuItem**

private javax.swing.JMenuItem **swBaseMenuItem**

Initiates the 'Software Base' event

93

Class caps.CAPSMain.EditMenu

```
java.lang.Object
   |
   +----java.awt.Component
         |
         +----java.awt.Container
               |
               +----javax.swing.JComponent
                     |
                     +----javax.swing.AbstractButton
                           |
                           +----
```

javax.swing.JMenuItem

```
               |
               +----
```

javax.swing.JMenu

```
   |
   +----
```

caps.CAPSMain.EditMenu

public class **EditMenu**

extends javax.swing.JMenu

implements java.awt.event.ActionListener

This class holds the 'Edit' menu items.

Constructors

◦DatabasesMenu

public **DatabasesMenu**()

Constructor for this class.

Methods

●actionPerformed

public void **actionPerformed**(java.awt.event.ActionEvent e)

Action event handler for the menu events.

Parameters:

e - The action event that is created by selecting a menu item from this menu

Variable Index

- adaMenuItem

  Initiates the 'Ada' event

- capsDefaultsMenuItem

  Initiates the 'CAPS Defaults' event

- changeReqMenuItem

  Initiates the 'Change Request' event

- hwModelMenuItem

  Initiates the 'Hardware Model' event

- interfaceMenuItem

  Initiates the 'Interface' event

- owner

  The main window which owns this menu.

- psdlMenuItem

  Initiates the 'PSDL' event

- requirementsMenuItem

  Initiates the 'Requirements' event

Constructor Index

- EditMenu(CAPSMainWindow)

  The constructor for this class.

Method Index

- actionPerformed(ActionEvent)

  Action event handler for the menu events.

Variables

- **psdlMenuItem**

  ```
  private javax.swing.JMenuItem psdlMenuItem
  ```

  Initiates the 'PSDL' event

- **adaMenuItem**

  ```
  private javax.swing.JMenuItem adaMenuItem
  ```

  Initiates the 'Ada' event

- **interfaceMenuItem**

  ```
  private javax.swing.JMenuItem interfaceMenuItem
  ```

  Initiates the 'Interface' event

- **requirementsMenuItem**

  ```
  private javax.swing.JMenuItem requirementsMenuItem
  ```

  Initiates the 'Requirements' event

- **changeReqMenuItem**

private `javax.swing.JMenuItem` **changeReqMenuItem**

Initiates the 'Change Request' event

● **capsDefaultsMenuItem**

private `javax.swing.JMenuItem` **capsDefaultsMenuItem**

Initiates the 'CAPS Defaults' event

● **hwModelMenuItem**

private `javax.swing.JMenuItem` **hwModelMenuItem**

Initiates the 'Hardware Model' event

● **owner**

protected `CAPSMainWindow` **owner**

The main window which owns this menu.

Constructors

◎ **EditMenu**

public **EditMenu**(`CAPSMainWindow` f)

The constructor for this class.

Parameters:

f - The parent class which has declared this menubar.

Methods

● **actionPerformed**

public void **actionPerformed**(`java.awt.event.ActionEvent` e)

Action event handler for the menu events.

Parameters:

e - The action event that is created by selecting a menu item from this menu

Class caps.CAPSMain.ExecSupportMenu

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----javax.swing.JComponent
                           |
                           +----javax.swing.AbstractButton
                                   |
                                   +----
                                       |
                                       +----
```

javax.swing.JMenuItem

javax.swing.JMenu

caps.CAPSMain.ExecSupportMenu

public class **ExecSupportMenu**

extends javax.swing.JMenu

implements java.awt.event.ActionListener

This class holds the 'Exec Support' menu items.

Variable Index

• compileMenuItem

  Initiates the 'Compile' event

• executeMenuItem

  Initiates the 'Execute' event

• scheduleMenuItem

  Initiates the 'Schedule' event

• translateMenuItem

  Initiates the 'Translate' event

Constructor Index

• ExecSupportMenu()

  Constructor for this class.

Method Index

• actionPerformed(ActionEvent)

  Action event handler for the menu events.

Variables

• translateMenuItem

Class caps.CAPSMain.ExitCAPSMain

```
java.lang.Object
    |
    +----java.awt.event.WindowAdapter
            |
            +----caps.CAPSMain.ExitCAPSMain
```

class ExitCAPSMain

extends java.awt.event.WindowAdapter

Closes the caps main window and exits from the program.

Variable Index

● capsMain

The main program that has declared this object

Constructor Index

● ExitCAPSMain(CAPSMainWindow)

The constructor for this class.

---

private javax.swing.JMenuItem **translateMenuItem**

Initiates the Translate' event

● **scheduleMenuItem**

private javax.swing.JMenuItem **scheduleMenuItem**

Initiates the 'Schedule' event

● **compileMenuItem**

private javax.swing.JMenuItem **compileMenuItem**

Initiates the 'Compile' event

● **executeMenuItem**

private javax.swing.JMenuItem **executeMenuItem**

Initiates the 'Execute' event

Constructors

○ **ExecSupportMenu**

public **ExecSupportMenu**()

Constructor for this class.

Methods

● **actionPerformed**

public void **actionPerformed**(java.awt.event.ActionEvent e)

Action event handler for the menu events.

Parameters:

e - The action event that is created by selecting a menu item from this menu

pressed.

Overrides:

windowClosing in class java.awt.event.WindowAdapter

Method Index

●windowClosing(WindowEvent)

Window event handler for the menu events.

Variables

●capsMain

CAPSMainWindow **capsMain**

The main program that has declared this object

Constructors

●ExitCAPSMain

public **ExitCAPSMain**(CAPSMainWindow caps)

The constructor for this class.

Parameters:

owner - The parent class which has declared this menubar.

Methods

●windowClosing

public void **windowClosing**(java.awt.event.WindowEvent e)

Window event handler for the menu events.

Parameters:

e - The window event that is created when the program close icon is

Class caps.CAPSMain.HelpMenu
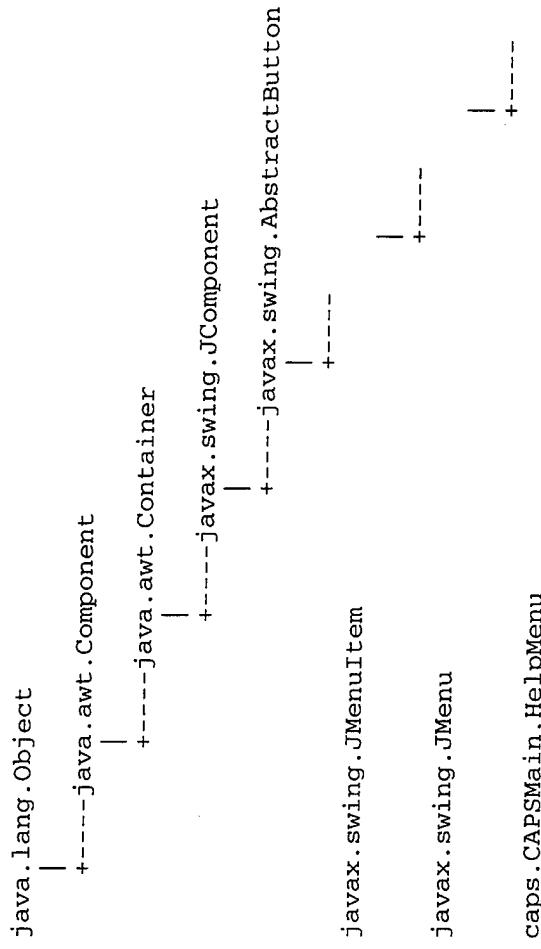
```
java.lang.Object
    |
    +----java.awt.Component
            |
            +----java.awt.Container
                    |
                    +----javax.swing.JComponent
                            |
                            +----javax.swing.AbstractButton
                                    |
                                    +----
```

```
javax.swing.JMenuItem
    |
    +----
```

```
javax.swing.JMenu
    |
    +----
```

```
caps.CAPSMain.HelpMenu
```

public class **HelpMenu**

extends javax.swing.JMenu

implements java.awt.event.ActionListener

This class implements the 'Help' menu.

Constructor Index

● HelpMenu()

Constructor for this class.

Method Index

● actionPerformed(ActionEvent)

Action event handler for the menu events.

Constructors

● **HelpMenu**

public **HelpMenu**()

Constructor for this class.

Methods

● **actionPerformed**

public void **actionPerformed**(java.awt.event.ActionEvent e)

Action event handler for the menu events.

Parameters:

e - The action event that is created by selecting a menu item from this menu

Class caps.CAPSMain.PrototypeMenu
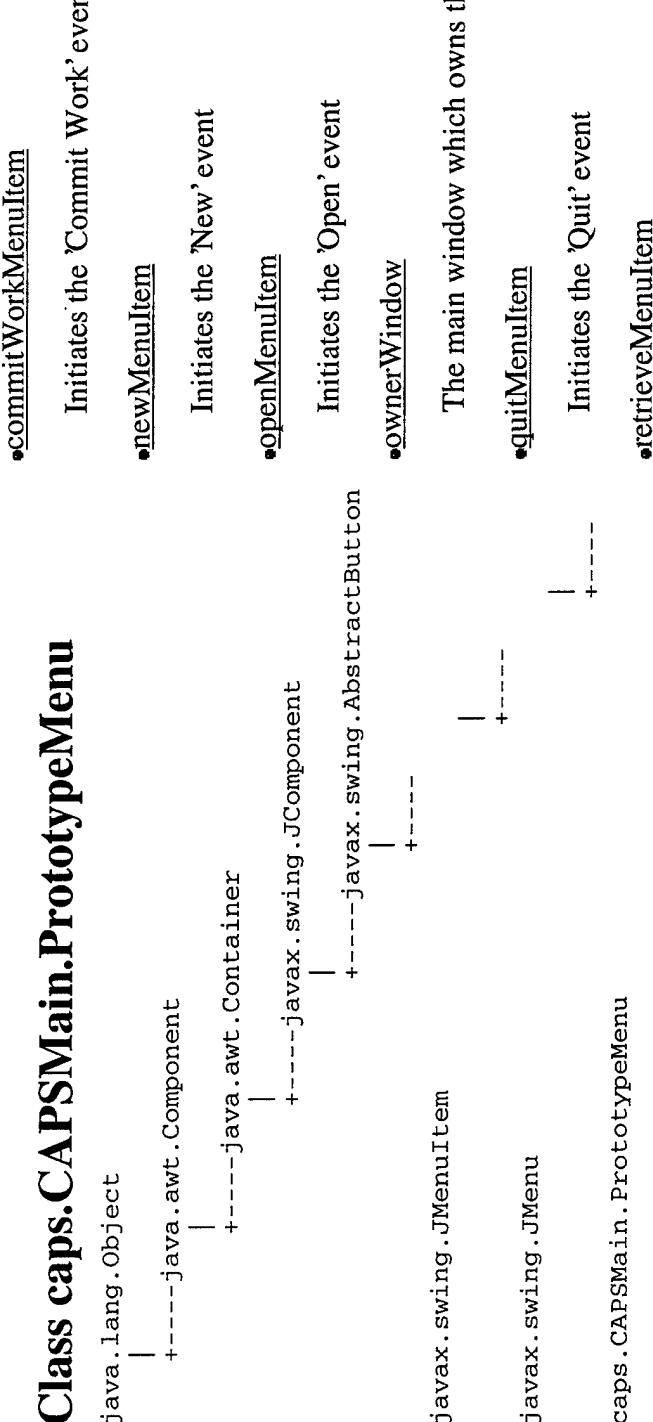
```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----javax.swing.JComponent
                           |
                           +----javax.swing.AbstractButton
                                   |
                                   +-----
                                           |
                                           +-----
                                                   |
                                                   +-----
```

javax.swing.JMenuItem

javax.swing.JMenu

caps.CAPSMain.PrototypeMenu

public class PrototypeMenu

extends javax.swing.JMenu

implements java.awt.event.ActionListener

This class holds the 'Prototype' menu items.

101

Variable Index

● commitWorkMenuItem

Initiates the 'Commit Work' event

● newMenuItem

Initiates the 'New' event

● openMenuItem

Initiates the 'Open' event

● ownerWindow

The main window which owns this menu.

● quitMenuItem

Initiates the 'Quit' event

● retrieveMenuItem

Initiates the 'Retrieve From DDB' event

Constructor Index

● PrototypeMenu(CAPSMainWindow)

Constructor for this class.

Method Index

● actionPerformed(ActionEvent)

Action event handler for the menu events.

● processNewMenuItem()

Handles the event which is caused by selecting the 'New' menu item.

● processOpenMenuItem()

Handles the event which is caused by selecting the 'Open' menu item.

Variables

● **newMenuItem**

private javax.swing.JMenuItem **newMenuItem**

Initiates the 'New' event

● **openMenuItem**

private javax.swing.JMenuItem **openMenuItem**

Initiates the 'Open' event

● **commitWorkMenuItem**

private javax.swing.JMenuItem **commitWorkMenuItem**

Initiates the 'Commit Work' event

● **retrieveMenuItem**

private javax.swing.JMenuItem **retrieveMenuItem**

Initiates the 'Retrieve From DDB' event

● **quitMenuItem**

private javax.swing.JMenuItem **quitMenuItem**

Initiates the 'Quit' event

● **ownerWindow**

protected CAPSMainWindow **ownerWindow**

The main window which owns this menu.

Constructors

⊙ **PrototypeMenu**

public **PrototypeMenu**(CAPSMainWindow owner)

Constructor for this class.

Parameters:

owner - The main window which has created this menu.

Methods

⊙ **actionPerformed**

public void **actionPerformed**(java.awt.event.ActionEvent e)

Action event handler for the menu events.

Parameters:

e - The action event that is created by selecting a menu item from this menu

⊙ **processNewMenuItem**

public void **processNewMenuItem**()

Handles the event which is caused by selecting the 'New' menu item.

⊙ **processOpenMenuItem**

public void **processOpenMenuItem**()

Handles the event which is caused by selecting the 'Open' menu

item.

Class caps.Display.DisplayComponent

```
java.lang.Object
    |
    +----caps.Display.DisplayComponent
```

public abstract class **DisplayComponent**

extends java.lang.Object

This is an abstract super class of EdgePath and DisplayVertex.

Variable Index

● dfc

  The DataFlowComponent that this object associates with.

● HANDLESIZE

  The size of the Handles.

● labelShape

  The shape of the label of the component.

● metShape

The shape of the met of the component .

Constructor Index

●DisplayComponent(DataFlowComponent)

The constructor is protected so it cannot be instantiated directly.

Method Index

●containsClickedPoint(int, int)

This abstract method is implemented in subclasses.

●delete()

This abstract method is implemented in subclasses.

●drawLabelShape(Graphics2D)

Gets the location of the label shape and draws it into the DrawPanel.

●drawMetShape(Graphics2D)

Gets the location of the met (or latency) shape and draws it into the DrawPanel.

●getDataFlowComponent()

Returns the DataFlowComponent that is associated with this object.

●getHandles()

This abstract method is implemented in subclasses.

●getLabelShapeBounds()

Returns the bounding rectangle of the label shape.

●getMetShapeBounds()

Returns the bounding rectangle of the met (or latency) shape.

●getShape()

This abstract method is implemented in subclasses.

●getStringHandles(Rectangle2D)

Creates a vector that holds the handles of a string (met or label).

●setLabelShape(Graphics2D)

Gets the label from the DataFlowComponent and creates a TextLayout shape for the label.

●setMetShape(Graphics2D)

Gets the met (or latency) from the DataFlowComponent and creates a TextLayout shape for the met.

●update()

This abstract method is implemented in subclasses.

Variables

●HANDLESIZE

public static final int **HANDLESIZE**

The size of the Handles.

●dfc

protected DataFlowComponent **dfc**

The DataFlowComponent that this object associates with.

labelShape

`java.awt.font.TextLayout` **`labelShape`**

The shape of the label of the component.

metShape

`java.awt.font.TextLayout` **`metShape`**

The shape of the met of the component.

Constructors

DisplayComponent

`protected` **`DisplayComponent`**(`DataFlowComponent` d)

The constructor is protected so it cannot be instantiated directly. param d the DataFlowComponent that is associated with this object.

Methods

getShape

`public abstract java.awt.Shape` **`getShape`**()

This abstract method is implemented in subclasses.

containsClickedPoint

`public abstract boolean` **`containsClickedPoint`**(`int xLoc,`
`int yLoc`)

This abstract method is implemented in subclasses.

getHandles

`public abstract java.util.Vector` **`getHandles`**()

This abstract method is implemented in subclasses.

update

`public abstract void` **`update`**()

This abstract method is implemented in subclasses.

delete

`public abstract void` **`delete`**()

This abstract method is implemented in subclasses.

setLabelShape

`public void` **`setLabelShape`**(`java.awt.Graphics2D g2D`)

Gets the label from the DataFlowComponent and creates a TextLayout shape for the label.

Parameters:

g2D - the graphics context of the DrawPanel

getLabelShapeBounds

`public java.awt.geom.Rectangle2D` **`getLabelShapeBounds`**()

Returns the bounding rectangle of the label shape.

Returns:

the bounding rectangle of the label shape.

drawLabelShape

`public void` **`drawLabelShape`**(`java.awt.Graphics2D g2D`)

Gets the location of the label shape and draws it into the DrawPanel.

Parameters:

g2D - the graphics context of the DrawPanel.

●getStringHandles

```
public java.util.Vector
getStringHandles(java.awt.geom.Rectangle2D r2D)
```

Creates a vector that holds the handles of a string (met or label).

Parameters:

r2D - the bounding rectangle of the string.

Returns:

returns the Vector that holds the handles.

●setMetShape

```
public void setMetShape(java.awt.Graphics2D g2D)
```

Gets the met (or latency) from the DataFlowComponent and creates a TextLayout shape for the met.

Parameters:

g2D - the graphics context of the DrawPanel

●getMetShapeBounds

```
public java.awt.geom.Rectangle2D getMetShapeBounds()
```

Returns the bounding rectangle of the met (or latency) shape.

Returns:

the bounding rectangle of the met (or latency) shape.

●drawMetShape

```
public void drawMetShape(java.awt.Graphics2D g2D)
```

Gets the location of the met (or latency) shape and draws it into the

DrawPanel.

Parameters:

g2D - the graphics context of the DrawPanel.

●getDataFlowComponent

```
public DataFlowComponent getDataFlowComponent()
```

Returns the DataFlowComponent that is associated with this object.

Returns:

the DataFlowComponent that is associated with this object.

Class caps.Display.DisplayExternal

```
java.lang.Object
    |
    +----caps.Display.DisplayComponent
            |
            +----caps.Display.DisplayExternal
```

public class **DisplayExternal**

extends DisplayComponent

An instance of this class is created when external streams are created.

Constructor Index

● DisplayExternal(External)

The constructor for this class.

Method Index

● containsClickedPoint(int, int)

Always returns false since the shape is not displayed in the DrawPanel.

● delete()

Deletes the external that is associated with this object.

● getHandles()

Returns the vector that contains the handles of the shape.

● getShape()

Returns the shape that represents the External.

● setLocation()

Sets the location of this shape on the DrawPanel

● update()

Updates the location and the width of this shape.

Variable Index

● external

The External object that is associated with this object.

● shape

The shape of the External.

Variables

● **external**

protected External **external**

The External object that is associated with this object.

● **shape**

protected java.awt.geom.Rectangle2D.Double **shape**

The shape of the External.

Constructors

○ **DisplayExternal**

public **DisplayExternal**(External e)

The constructor for this class. param e the External that is associated with this object.

Methods

● **setLocation**

public void **setLocation**()

Sets the location of this shape on the DrawPanel

● **update**

public void **update**()

Updates the location and the width of this shape.

Overrides:

update in class DisplayComponent

● **containsClickedPoint**

public boolean **containsClickedPoint**(int xLoc,
                                        int yLoc)

Always returns false since the shape is not displayed in the DrawPanel.

Parameters:

xLoc - the x location of the clicked point.

yLoc - the y location of the clicked point.

Returns:

false.

Overrides:

containsClickedPoint in class DisplayComponent

● **getHandles**

public java.util.Vector **getHandles**()

Returns the vector that contains the handles of the shape.

Returns:

an empty Vector.

Overrides:

getHandles in class DisplayComponent

● **getShape**

public java.awt.Shape **getShape**()

Returns the shape that represents the External.

Class caps.Display.DisplayVertex

```
java.lang.Object
   |
   +----caps.Display.DisplayComponent
           |
           +----caps.Display.DisplayVertex
```

public class **DisplayVertex**

extends DisplayComponent

This class holds a shape for its associated Vertex. It can either be a rectangle for terminators or it can be a circle for the operators.

Variable Index

●shape

The shape of the Vertex.

●vertex

The Vertex that is associated with this object.

Returns:

the shape that represents the External.

Overrides:

getShape in class DisplayComponent

●**delete**

public void **delete**()

Deletes the external that is associated with this object.

Overrides:

delete in class DisplayComponent

Constructor Index

● DisplayVertex(Vertex)

The constructor for this class.

Method Index

● containsClickedPoint(int, int)

Checks whether the bounding box of the shape contains the the location where the mouse is clicked.

● delete()

Deletes the vertex that is associated with this object.

● getHandles()

Returns the vector that contains the handles of the shape.

● getInnerShape()

This method is called if the Vertex is composite.

● getPaintedShape()

Returns a shape that is slightly smaller than the shape of this object.

● getShape()

Returns the shape that represents the Vertex.

● setLocation()

Sets the location of this shape on the DrawPanel

● setShape()

Sets the shape of this object to a circle if the associated Vertex is an operator or sets it to a rectangle if the Vertex is a Terminator

● setWidth()

Sets the width of this shape.

● update()

Updates the location and the width of this shape.

Variables

● vertex

protected Vertex **vertex**

The Vertex that is associated with this object.

● shape

protected java.awt.geom.RectangularShape **shape**

The shape of the Vertex.

Constructors

○ DisplayVertex

public **DisplayVertex**(Vertex v)

The constructor for this class. param v the Vertex that is associated with this object.

Methods

●setLocation

```
public void setLocation()
```

Sets the location of this shape on the DrawPanel

●setWidth

```
public void setWidth()
```

Sets the width of this shape.

●update

```
public void update()
```

Updates the location and the width of this shape.

Overrides:

update in class <u>DisplayComponent</u>

●containsClickedPoint

```
public boolean containsClickedPoint(int xLoc,
                                    int yLoc)
```

Checks whether the bounding box of the shape contains the the location where the mouse is clicked.

Parameters:

xLoc - the x location of the clicked point.

yLoc - the y location of the clicked point.

Returns:

true if the bounding box contains the clicked point.

Overrides:

containsClickedPoint in class <u>DisplayComponent</u>

●getHandles

```
public java.util.Vector getHandles()
```

Returns the vector that contains the handles of the shape.

Returns:

the vector that contains the handles of the shape

Overrides:

getHandles in class <u>DisplayComponent</u>

●setShape

```
public void setShape()
```

Sets the shape of this object to a circle if the associated Vertex is an operator or sets it to a rectangle if the Vertex is a Terminator

●getShape

```
public java.awt.Shape getShape()
```

Returns the shape that represents the Vertex.

Returns:

the shape that represents the Vertex.

Overrides:

getShape in class <u>DisplayComponent</u>

●getInnerShape

```
public java.awt.Shape getInnerShape()
```

Class caps.Display.EdgePath

```
java.lang.Object
    |
    +----caps.Display.DisplayComponent
            |
            +----caps.Display.EdgePath
```

public class EdgePath

extends DisplayComponent

This class represents an Edge on the DrawPanel. It contains a GeneralPath shape to represent the Edge.

Variable Index

●edge

   The Edge that is associated with this object.

●shape

   The shape of the Edge.

This method is called if the Vertex is composite. It calculates and returns a smaller inner shape.

Returns:

   the inner shape for the composite Vertex.

●**getPaintedShape**

public java.awt.Shape **getPaintedShape**()

   Returns a shape that is slightly smaller than the shape of this object. The shape that is returned will be painted with the color of the Vertex.

Returns:

   a shape that is slightly smaller than the shape of th object.

●**delete**

public void **delete**()

   Deletes the vertex that is associated with this object.

Overrides:

   delete in class DisplayComponent

Constructor Index

● EdgePath(Edge)

    The constructor for this class.

Method Index

● buildArrowHead(Point, Point)

    Creates an arrow head for the stream.

● containsClickedPoint(int, int)

    Checks whether the bounding box of the shape contains the the location where the mouse is clicked.

● delete()

    Deletes the Edge that is associated with this object.

● getHandles()

    Returns the vector that contains the handles of the shape.

● getShape()

    Returns the shape that represents the Edge.

● update()

    Updates the shape by polling values from the associated Edge object.

113

Variables

● edge

protected Edge **edge**

    The Edge that is associated with this object.

● shape

protected java.awt.geom.GeneralPath **shape**

    The shape of the Edge.

Constructors

● EdgePath

public **EdgePath**(Edge e)

    The constructor for this class. param e the Edge that is associated with this object.

Methods

● getShape

public java.awt.Shape **getShape**()

    Returns the shape that represents the Edge.

Returns:

    the shape that represents the Edge.

Overrides:

    getShape in class DisplayComponent

●**containsClickedPoint**
```
public boolean containsClickedPoint(int xLoc,
                                    int yLoc)
```

Checks whether the bounding box of the shape contains the the location where the mouse is clicked.

Parameters:

xLoc - the x location of the clicked point.

yLoc - the y location of the clicked point.

Returns:

true if the bounding box contains the clicked point.

Overrides:

containsClickedPoint in class DisplayComponent

●**update**
```
public void update()
```

Updates the shape by polling values from the associated Edge object.

Overrides:

update in class DisplayComponent

●**buildArrowHead**
```
public void buildArrowHead(java.awt.Point last,
                           java.awt.Point end)
```

Creates an arrow head for the stream.

Parameters:

last - the point before the ending point of the stream.

end - the last pointof the stream.

●**getHandles**
```
public java.util.Vector getHandles()
```

Returns the vector that contains the handles of the shape.

Returns:

the vector that contains the handles of the shape

Overrides:

getHandles in class DisplayComponent

●**delete**
```
public void delete()
```

Deletes the Edge that is associated with this object.

Overrides:

delete in class DisplayComponent

114

Class caps.GraphEditor.ColorConstants

java.lang.Object
|
+----caps.GraphEditor.ColorConstants

public class **ColorConstants**

extends java.lang.Object

Variable Index

● COLOR_NAMES

● RGB_VALUES

Constructor Index

● ColorConstants()

Variables

● COLOR_NAMES

public static java.lang.String[] **COLOR_NAMES**

● RGB_VALUES

public static int[] **RGB_VALUES**

Constructors

○ ColorConstants

public **ColorConstants**()

115

Class caps.GraphEditor.DrawPanel

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----javax.swing.JComponent
                           |
                           +----javax.swing.JPanel
                                   |
                                   +----
```

caps.GraphEditor.DrawPanel

public class **DrawPanel**

extends javax.swing.JPanel

implements java.awt.event.MouseListener,
java.awt.event.MouseMotionListener, java.awt.event.ActionListener

The drawpanel is the place where the prototypes are drawn on the screen.

Variable Index

- **bounds**

- **currentColor**

- **currentComponent**

  Current component is either an OPERATOR, or a TERMINATOR or a STREAM according to the selection from the toolbar.

- **currentEdge**

- **currentFont**

- **DEFAULT_CURSOR**

- **diagonalPoint**

- **displayComponentVector**

  This vector holds the shapes that are drawn in the DrawPanel.

- **ePropertyPanel**

- **HAND_CURSOR**

- handlesVector

- HEIGHT

  The constant height of the DrawPanel

- IS_COLLECTING_POINTS

- MOVE_CURSOR

- MOVING_ALL

- MOVING_COMPONENT

- MOVING_LABEL

- MOVING_MET

- OPERATOR

  The constant which specifies an operator

- parentFrame

  The frame which has created this DrawPanel object

- parentVertex

- popupMenu

- prevPoint

- RESIZING

- selectAllMode

- selectedComponent

- selectionDefault

- selectMode

  The value of this variable is true if the toolbar is in the select mode

- STREAM

  The constant which specifies a stream

- TERMINATOR

  The constant which specifies a terminator

117

- vPropertyPanel

- WIDTH

  The constant width of the DrawPanel

Constructor Index

- DrawPanel(Editor, Vertex)

  Constructs a new ToolBar object

Method Index

- actionPerformed(ActionEvent)

- changeLevel(Vertex)

- clearAllComponentsFromScreen(Graphics2D)

- decompose()

- deleteSelectedComponent()

- eraseHandles()

- getDiagonalPoint(Rectangle2D)

- getParentVertex()

- getPreferredSize()

  Sets the size of the panel to WIDTH and HEIGHT

- gotoParent()

- gotoRoot()

- isHoldingHandle(int, int)

- mouseClicked(MouseEvent)

  Handles the event that occurs when a mouse button is pressed on this panel

- mouseDragged(MouseEvent)

  Handles the event that occurs when the mouse is dragged on this panel

118

- **mouseEntered(MouseEvent)**

  Handles the event that occurs when the mouse enters into the panel.

- **mouseExited(MouseEvent)**

  Handles the event that occurs when the mouse exits the panel.

- **mouseMoved(MouseEvent)**

  Handles the event that occurs when the mouse is moved on this panel

- **mousePressed(MouseEvent)**

  Handles the event that occurs when a mouse button is clicked on this panel

- **mouseReleased(MouseEvent)**

  Handles the event that occurs when a mouse button is released on this panel

- **paint(Graphics)**

  This method is called to repaint all the components when necessary.

- **paintComponent(DisplayComponent)**

  Paints the component into this panel by calling the graphics2D.draw(Shape) method.

- **processOperator(int, int)**

  Creates a new Operator and a new OperatorCircle object.

- **processStream(int, int, int)**

- **processTerminator(int, int)**

  Creates a new Terminator and a new TerminatorRectangle object.

- **rubberBandLine(int, int)**

- **selectAllComponents()**

- **setCurrentColor(int)**

- **setCurrentComponent(int)**

  Sets the currentComponent variable to the specified argument.

- **setCurrentFont(int)**

- **setMenuBarItems()**

- **setParentVertex(Vertex, Graphics2D)**

- **setSelectAllMode(boolean)**

- **setSelectedDFC(DataFlowComponent)**

119

- •setSelectionDefault(boolean)

- •setSelectMode(boolean)

    Sets the select mode to true or false.

- •showProperties(DisplayComponent)

Variables

- •WIDTH

    `public static final int WIDTH`

    The constant width of the DrawPanel

- •HEIGHT

    `public static final int HEIGHT`

    The constant height of the DrawPanel

- •DEFAULT_CURSOR

    `private final java.awt.Cursor DEFAULT_CURSOR`

- •HAND_CURSOR

    `private final java.awt.Cursor HAND_CURSOR`

- •MOVE_CURSOR

    `private final java.awt.Cursor MOVE_CURSOR`

- •OPERATOR

    `public static final int OPERATOR`

    The constant which specifies an operator

- •TERMINATOR

    `public static final int TERMINATOR`

    The constant which specifies a terminator

- •STREAM

    `public static final int STREAM`

    The constant which specifies a stream

- •selectMode

    `protected boolean selectMode`

    The value of this variable is true if the toolbar is in the select mode

- •parentFrame

    `protected Editor parentFrame`

    The frame which has created this DrawPanel object

- •displayComponentVector

    `protected java.util.Vector displayComponentVector`

    This vector holds the shapes that are drawn in the DrawPanel. Each shape is redrawn in the paint method by polling them from this Vector.

- •handlesVector

    `protected java.util.Vector handlesVector`

- •selectedComponent

    `protected DisplayComponent selectedComponent`

- •MOVING_COMPONENT

    `protected boolean MOVING_COMPONENT`

- •MOVING_LABEL

    `protected boolean MOVING_LABEL`

- •MOVING_MET

protected boolean **MOVING_MET**

●**RESIZING**

protected boolean **RESIZING**

●**IS_COLLECTING_POINTS**

protected boolean **IS_COLLECTING_POINTS**

●**MOVING_ALL**

protected boolean **MOVING_ALL**

●**diagonalPoint**

protected java.awt.geom.Point2D **diagonalPoint**

●**vPropertyPanel**

protected VertexProperties **vPropertyPanel**

●**ePropertyPanel**

protected EdgeProperties **ePropertyPanel**

●**parentVertex**

protected Vertex **parentVertex**

●**currentEdge**

protected EdgePath **currentEdge**

●**selectionDefault**

protected boolean **selectionDefault**

●**currentComponent**

protected int **currentComponent**

Current component is either an OPERATOR, or a TERMINATOR or a STREAM according to the selection from the toolbar.

●**popupMenu**

protected Popup **popupMenu**

●**selectAllMode**

protected boolean **selectAllMode**

●**prevPoint**

protected java.awt.Point **prevPoint**

●**bounds**

protected java.awt.Rectangle **bounds**

●**currentColor**

protected int **currentColor**

●**currentFont**

protected int **currentFont**

Constructors

●**DrawPanel**

public **DrawPanel**(Editor frame, Vertex root)

Constructs a new ToolBar object

Parameters:

frame - The parent frame of this DrawPanel object.

Methods

●**setSelectMode**

public void **setSelectMode**(boolean mode)

Sets the select mode to true or false. The panel is generally in the select mode unless another button is pressed in the toolbar.

Parameters:

mode - true if the panel is going to be in the select mode.

●**gotoRoot**

public void **gotoRoot** ()

●**gotoParent**

public void **gotoParent** ()

●**decompose**

public void **decompose** ()

●**changeLevel**

public void **changeLevel** (Vertex parent)

●**setParentVertex**

public void **setParentVertex** (Vertex v,
                          java.awt.Graphics2D g2D)

●**eraseHandles**

public void **eraseHandles** ()

●**clearAllComponentsFromScreen**

public void
clearAllComponentsFromScreen (java.awt.Graphics2D g2D)

●**setCurrentComponent**

public void **setCurrentComponent** (int component)

Sets the currentComponent variable to the specified argument.

Parameters:

component - OPERATOR, TERMINATOR or STREAM

●**setSelectedDFC**

public void **setSelectedDFC** (DataFlowComponent dfc)

●**processOperator**

public void **processOperator** (int xLoc,
                          int yLoc)

Creates a new Operator and a new OperatorCircle object. Calls the paintComponent () method to draw the component to this panel.

Parameters:

xLoc - The x location of the component.

yLoc - The y location of the component.

●**processTerminator**

public void **processTerminator** (int xLoc,
                            int yLoc)

Creates a new Terminator and a new TerminatorRectangle object. Calls the paintComponent () method to draw the component to this panel.

Parameters:

xLoc - The x location of the component.

yLoc - The y location of the component.

●**processStream**

public void **processStream** (int x,
                        int y,
                        int clicks)

●**paintComponent**

public void **paintComponent** (DisplayComponent component)

Paints the component into this panel by calling the graphics2D.draw(Shape) method.

Parameters:

component - The component to be drawn into the panel

●paint

`public void ` **`paint`**`(java.awt.Graphics g)`

This method is called to repaint all the components when necessary.

Parameters:

g - The graphics context of the panel

Overrides:

paint in class javax.swing.JComponent

●getPreferredSize

`public java.awt.Dimension ` **`getPreferredSize`**`()`

Sets the size of the panel to WIDTH and HEIGHT

Returns:

Returns a new Dimension object initialized to the WIDTH and HEIGHT parameters.

Overrides:

getPreferredSize in class javax.swing.JComponent

●mousePressed

`public void ` **`mousePressed`**`(java.awt.event.MouseEvent e)`

Handles the event that occurs when a mouse button is clicked on this panel

Parameters:

e - The MouseEvent that occurs.

123

●setMenuBarItems

`public void ` **`setMenuBarItems`**`()`

●mouseEntered

`public void ` **`mouseEntered`**`(java.awt.event.MouseEvent e)`

Handles the event that occurs when the mouse enters into the panel.

Parameters:

e - The MouseEvent that occurs.

●mouseExited

`public void ` **`mouseExited`**`(java.awt.event.MouseEvent e)`

Handles the event that occurs when the mouse exits the panel.

Parameters:

e - The MouseEvent that occurs.

●mouseClicked

`public void ` **`mouseClicked`**`(java.awt.event.MouseEvent e)`

Handles the event that occurs when a mouse button is pressed on this panel

Parameters:

e - The MouseEvent that occurs.

●showProperties

`public void ` **`showProperties`**`(DisplayComponent d)`

●mouseReleased

`public void ` **`mouseReleased`**`(java.awt.event.MouseEvent e)`

Handles the event that occurs when a mouse button is released on

this panel

Parameters:

e - The MouseEvent that occurs.

●**mouseDragged**

`public void mouseDragged(java.awt.event.MouseEvent e)`

Handles the event that occurs when the mouse is dragged on this panel

Parameters:

e - The MouseEvent that occurs.

●**isHoldingHandle**

`public boolean isHoldingHandle(int x,`
`                               int y)`

●**getDiagonalPoint**

`public java.awt.geom.Point2D`
`getDiagonalPoint(java.awt.geom.Rectangle2D rect)`

●**mouseMoved**

`public void mouseMoved(java.awt.event.MouseEvent e)`

Handles the event that occurs when the mouse is moved on this panel

Parameters:

e - The MouseEvent that occurs.

●**actionPerformed**

`public void actionPerformed(java.awt.event.ActionEvent e)`

●**deleteSelectedComponent**

124

`public void deleteSelectedComponent()`

●**getParentVertex**

`public Vertex getParentVertex()`

●**setSelectAllMode**

`public void setSelectAllMode(boolean b)`

●**selectAllComponents**

`public void selectAllComponents()`

●**setSelectionDefault**

`public void setSelectionDefault(boolean b)`

●**setCurrentColor**

`public void setCurrentColor(int colorIndex)`

●**setCurrentFont**

`public void setCurrentFont(int fontIndex)`

○**rubberBandLine**

`protected void rubberBandLine(int x,`
`                              int y)`

Class caps.GraphEditor.EdgeProperties

```
java.lang.Object
    |
    +----java.awt.Component
            |
            +----java.awt.Container
                    |
                    +----java.awt.Window
                            |
                            +----java.awt.Dialog
                                    |
                                    +----
                                            |
                                            +----
```

```
javax.swing.JDialog
```

```
caps.GraphEditor.EdgeProperties
```

public class EdgeProperties

extends javax.swing.JDialog

implements java.awt.event.ActionListener

Variable Index

● cancelButton

● ePath

● helpButton

● initValueButton

● initValueField

● latencyField

● latencyUnitsCombo

● nameField

● noButton

● okButton

● parentFrame

125

- setEdgePath(EdgePath)

- showErrorDialog(String)

Variables

- **targetEdge**
  Edge **targetEdge**

- **ePath**
  EdgePath **ePath**

- **nameField**
  javax.swing.JTextField **nameField**

- **streamTypeField**
  javax.swing.JTextField **streamTypeField**

- **latencyField**
  javax.swing.JTextField **latencyField**

- **initValueField**
  javax.swing.JTextField **initValueField**

- **noButton**
  javax.swing.JRadioButton **noButton**

- **yesButton**
  javax.swing.JRadioButton **yesButton**

- **latencyUnitsCombo**

- streamTypeField

- targetEdge

- yesButton

Constructor Index

- EdgeProperties(Editor)

Method Index

- actionPerformed(ActionEvent)

- getUnitsCombo()

- initialize()

- setEdge(Edge)

javax.swing.JComboBox **latencyUnitsCombo**

● **okButton**

javax.swing.JButton **okButton**

● **cancelButton**

javax.swing.JButton **cancelButton**

● **helpButton**

javax.swing.JButton **helpButton**

● **initValueButton**

javax.swing.JButton **initValueButton**

● **parentFrame**

Editor **parentFrame**

Constructors

○ **EdgeProperties**

public **EdgeProperties**(Editor parent)

Methods

● **initialize**

public void **initialize**()

● **setEdge**

public void **setEdge**(Edge e)

● **setEdgePath**

public void **setEdgePath**(EdgePath e)

● **actionPerformed**

public void **actionPerformed**(java.awt.event.ActionEvent e)

● **showErrorDialog**

public void **showErrorDialog**(java.lang.String str)

● **getUnitsCombo**

public javax.swing.JComboBox **getUnitsCombo**()

127

Class caps.GraphEditor.Editor

```
java.lang.Object
   |
   +----java.awt.Component
          |
          +----java.awt.Container
                 |
                 +----java.awt.Window
                        |
                        +----java.awt.Frame
                               |
                               +----javax.swing.JFrame
                                      |
                                      +----
```

caps.GraphEditor.Editor

public class **Editor**

extends javax.swing.JFrame

implements java.lang.Runnable

The main frame for the Graph Editor. It constructs and drives the other features.

Variable Index

● drawPanel

    The panel that the drawing operations are performed.

● INITIAL_HEIGHT

    The initial height of the Graph Editor

● INITIAL_WIDTH

    The initial width of the GraphEditor

● innerSplit

    Includes the treePanel and the drawPanel.

● panel

    The panel that includes the Drawing area and tree view

● prototypeFile

● root

● saveRequired

● statusBar

- tBar

  the main toolbar of the GraphEditor

- treePanel

  The panel that includes the tree structure to view

- types

Constructor Index

- Editor(File, Vertex, DataTypes)

  The constructor for the editor frame

Method Index

- checkSaved()

- getDataTypes()

- getDrawPanel()

  *** Pending -- is it needed? *** Returns the DrawPanel object in this frame

- getPrototypeFile()

- getRoot()

- getSplitPane()

- getStatusBar()

- getToolBar()

  *** Pending -- is it needed? *** Returns the toolBar object in this frame

- getTreePanel()

  *** Pending -- is it needed? *** Returns the TreePanel object in this frame

- initialize()

  The initialization of the GUI takes place here

- isSaveRequired()

- run()

- savePrototype()

129

•setSaveRequired(boolean)

Variables

•**panel**

protected javax.swing.JPanel **panel**

The panel that includes the Drawing area and tree view

•**innerSplit**

protected javax.swing.JSplitPane **innerSplit**

Includes the treePanel and the drawPanel.

•**treePanel**

protected TreePanel **treePanel**

The panel that includes the tree structure to view

•**drawPanel**

protected DrawPanel **drawPanel**

The panel that the drawing operations are performed.

•**statusBar**

protected StatusBar **statusBar**

•**tBar**

protected ToolBar **tBar**

the main toolbar of the GraphEditor

•**root**

protected Vertex **root**

•**INITIAL_WIDTH**

private final int **INITIAL_WIDTH**

The initial width of the GraphEditor

•**INITIAL_HEIGHT**

private final int **INITIAL_HEIGHT**

The initial height of the Graph Editor

•**types**

protected DataTypes **types**

•**prototypeFile**

protected java.io.File **prototypeFile**

•**saveRequired**

protected boolean **saveRequired**

Constructors

•**Editor**

public **Editor**(java.io.File prototype, Vertex r, DataTypes t)

The constructor for the editor frame

Methods

•**run**

public void **run**()

•**initialize**

public void **initialize**()

The initialization of the GUI takes place here

●**getTreePanel**

public TreePanel **getTreePanel**()

*** Pending -- is it needed? *** Returns the TreePanel object in this frame

Returns:

the treePanel object in this JFrame

●**getDrawPanel**

public DrawPanel **getDrawPanel**()

*** Pending -- is it needed? *** Returns the DrawPanel object in this frame

Returns:

the drawPanel object in this JFrame

●**getToolBar**

public ToolBar **getToolBar**()

*** Pending -- is it needed? *** Returns the toolBar object in this frame

Returns:

the toolBar object in this JFrame

●**getStatusBar**

public StatusBar **getStatusBar**()

●**getSplitPane**

public javax.swing.JSplitPane **getSplitPane**()

●**getRoot**

public Vertex **getRoot**()

●**getDataTypes**

public DataTypes **getDataTypes**()

●**getPrototypeFile**

public java.io.File **getPrototypeFile**()

●**setSaveRequired**

public void **setSaveRequired**(boolean b)

●**isSaveRequired**

public boolean **isSaveRequired**()

●**checkSaved**

public boolean **checkSaved**()

●**savePrototype**

public void **savePrototype**()

Class caps.GraphEditor.EditorMenuBar

```
java.lang.Object
  |
  +----java.awt.Component
          |
          +----java.awt.Container
                  |
                  +----javax.swing.JComponent
                          |
                          +----javax.swing.JMenuBar
                                  |
                                  +----
```

caps.GraphEditor.EditorMenuBar

public class **EditorMenuBar**

extends javax.swing.JMenuBar

The MenuBar of the Graph Editor.

Constructor Index

●EditorMenuBar(Editor)

The constructor for this class.

Constructors

●**EditorMenuBar**

public **EditorMenuBar**(Editor parent)

The constructor for this class.

Class caps.GraphEditor.ExitEditor

```
java.lang.Object
    |
    +----java.awt.event.WindowAdapter
            |
            +----caps.GraphEditor.ExitEditor
```

class ExitEditor

extends java.awt.event.WindowAdapter

Closes the caps main window and exits from the program.

Variable Index

● editor

Constructor Index

● ExitEditor(Editor)

Method Index

● windowClosing(WindowEvent)

Window event handler for the menu events.

Variables

● **editor**

Editor **editor**

Constructors

● **ExitEditor**

public **ExitEditor**(Editor e)

Methods

● **windowClosing**

public void **windowClosing**(java.awt.event.WindowEvent e)

Window event handler for the menu events.

Parameters:

e – The window event that is created when the program close icon is pressed.

Overrides:

133

Class caps.GraphEditor.FontConstants

```
java.lang.Object
   |
   +----caps.GraphEditor.FontConstants
```

public class **FontConstants**

extends java.lang.Object

Variable Index

● FONT_NAMES

● FONT_VALUES

Constructor Index

● FontConstants()

Variables

● FONT_VALUES

Class caps.GraphEditor.GE_EditMenu

```
java.lang.Object
    |
    +----java.awt.Component
            |
            +----java.awt.Container
                    |
                    +----javax.swing.JComponent
                            |
                            +----javax.swing.AbstractButton
                                    |
                                    +----
                                            |
                                            +----
```

javax.swing.JMenuItem

javax.swing.JMenu

caps.GraphEditor.GE_EditMenu

public class GE_EditMenu

extends javax.swing.JMenu

implements java.awt.event.ActionListener

Constructs the Edit menu of the menubar. Also handles the events
associated with the Edit Menu.

---

public static java.lang.String[] **FONT_VALUES**

● **FONT_NAMES**

public static java.lang.String[] **FONT_NAMES**

Constructors

◉ **FontConstants**

public **FontConstants**()

Variables

●undoMenuItem

private javax.swing.JMenuItem **undoMenuItem**

　　Initiates the 'Undo' event

●redoMenuItem

private javax.swing.JMenuItem **redoMenuItem**

　　Initiates the 'Redo' event

●selectAllMenuItem

private javax.swing.JMenuItem **selectAllMenuItem**

　　Initiates the 'Paste' event

●deleteMenuItem

private javax.swing.JMenuItem **deleteMenuItem**

　　Initiates the 'Delete' event

●parent

private Editor **parent**

Constructors

○GE_EditMenu

public **GE_EditMenu**(Editor e)

　　The constructor for the Edit menu

Variable Index

●deleteMenuItem

　　Initiates the 'Delete' event

●parent

●redoMenuItem

　　Initiates the 'Redo' event

●selectAllMenuItem

　　Initiates the 'Paste' event

●undoMenuItem

　　Initiates the 'Undo' event

Constructor Index

○GE_EditMenu(Editor)

　　The constructor for the Edit menu

Method Index

●actionPerformed(ActionEvent)

　　Handles the menu events that occur when one of the menu items is
　　selected

Class caps.GraphEditor.GE_FileMenu

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----javax.swing.JComponent
                           |
                           +----javax.swing.AbstractButton
```

```
javax.swing.JMenuItem
```

```
javax.swing.JMenu
```

```
caps.GraphEditor.GE_FileMenu
```

public class GE_FileMenu

extends javax.swing.JMenu

implements java.awt.event.ActionListener

Constructs the File menu of the menubar. Also handles the events associated with the File Menu.

Methods

● **actionPerformed**

public void **actionPerformed**(java.awt.event.ActionEvent e)

Handles the menu events that occur when one of the menu items is selected

Parameters:

e - The associated ActionEvent

137

Variables

●saveMenuItem

`private javax.swing.JMenuItem saveMenuItem`

    Initiates the 'Save' event

●restoreMenuItem

`private javax.swing.JMenuItem restoreMenuItem`

    Initiates the 'Restore From Save' event

●printMenuItem

`private javax.swing.JMenuItem printMenuItem`

    Initiates the 'Print' event

●exitMenuItem

`private javax.swing.JMenuItem exitMenuItem`

    Initiates the 'Exit' event

●parent

`private Editor parent`

Constructors

●GE_FileMenu

`public GE_FileMenu(Editor e)`
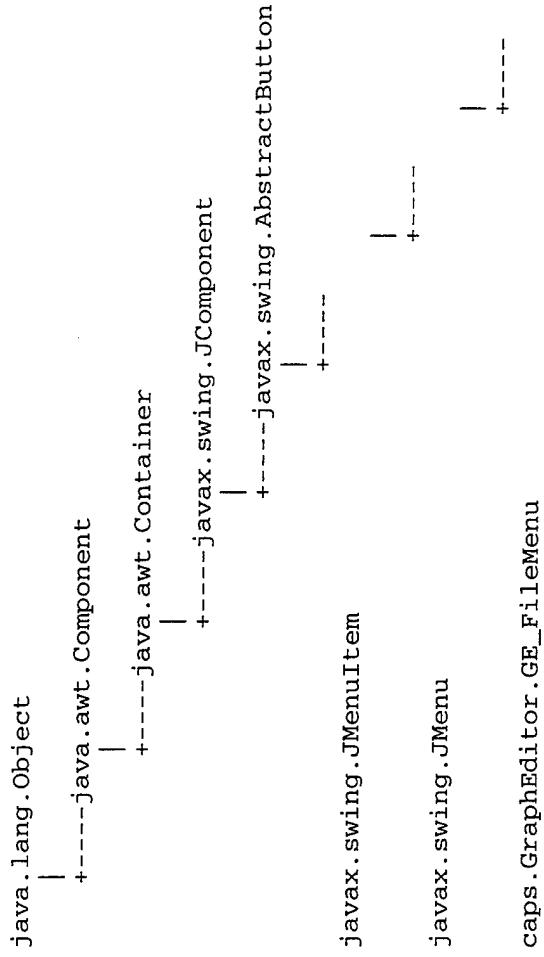
    The constructor for the File menu

Variable Index

●exitMenuItem

    Initiates the 'Exit' event

●parent

●printMenuItem

    Initiates the 'Print' event

●restoreMenuItem

    Initiates the 'Restore From Save' event

●saveMenuItem

    Initiates the 'Save' event

Constructor Index

●GE_FileMenu(Editor)

    The constructor for the File menu

Method Index

●actionPerformed(ActionEvent)

    Handles the menu events that occur when one of the menu items is selected

138

Class caps.GraphEditor.GE_HelpMenu

```
java.lang.Object
  |
  +----java.awt.Component
          |
          +----java.awt.Container
                  |
                  +----javax.swing.JComponent
                          |
                          +----javax.swing.AbstractButton
                                  |
                                  +-----
                                          |
                                          +-----
                                                  |
                                                  +-----
```

javax.swing.JMenuItem

javax.swing.JMenu

caps.GraphEditor.GE_HelpMenu

public class **GE_HelpMenu**

extends javax.swing.JMenu

implements java.awt.event.ActionListener

Constructs the Help menu of the menubar. Also handles the events associated with the Help Menu.

Methods

●actionPerformed

public void **actionPerformed**(java.awt.event.ActionEvent e)

Handles the menu events that occur when one of the menu items is selected

Parameters:

e - The associated ActionEvent

139

Variable Index

Constructor Index

Method Index

140

Variables

• **psdlGrammarMenuItem**

private javax.swing.JMenuItem **psdlGrammarMenuItem**

    Initiates the 'PSDL Grammar' event

• **operatorsMenuItem**

private javax.swing.JMenuItem **operatorsMenuItem**

    Initiates the 'Operators' event

• **streamsMenuItem**

private javax.swing.JMenuItem **streamsMenuItem**

    Initiates the 'Streams' event

• **exceptionsMenuItem**

private javax.swing.JMenuItem **exceptionsMenuItem**

    Initiates the 'Exceptions' event

• **timersMenuItem**

private javax.swing.JMenuItem **timersMenuItem**

    Initiates the 'Timers' event

Constructors

• **GE_HelpMenu**

public **GE_HelpMenu**(Editor e)

    The constructor for the Help menu

Class caps.GraphEditor.GE_PSDLMenu

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----javax.swing.JComponent
                           |
                           +----javax.swing.AbstractButton
                                   |
                                   +----
                                           |
                                           +----
                                                   |
                                                   +----
```

javax.swing.JMenuItem

javax.swing.JMenu

caps.GraphEditor.GE_PSDLMenu

public class GE_PSDLMenu

extends javax.swing.JMenu

implements java.awt.event.ActionListener

Constructs the PSDL menu of the menubar. Also handles the events associated with the PSDL Menu.

Methods

● **actionPerformed**

`public void actionPerformed(java.awt.event.ActionEvent e)`

Handles the menu events that occur when one of the menu items is selected

Parameters:

e - The associated ActionEvent

141

Variable Index

Constructor Index

Method Index

Variables

- **gotoRootMenuItem**

  `private javax.swing.JMenuItem gotoRootMenuItem`

  Initiates the 'Goto Root' event

- **gotoParentMenuItem**

  `private javax.swing.JMenuItem gotoParentMenuItem`

  Initiates the 'Goto Parent' event

- **decomposeMenuItem**

  `private javax.swing.JMenuItem decomposeMenuItem`

  Initiates the 'Decompose' event

- **parent**

  `private Editor parent`

Constructors

- **GE_PSDLMenu**

  `public GE_PSDLMenu(Editor e)`

  The constructor for the PSDL menu

Methods

- **actionPerformed**

  `public void actionPerformed(java.awt.event.ActionEvent e)`

  Handles the menu events that occur when one of the menu items is selected

  Parameters:

  e - The associated ActionEvent

142

Class caps.GraphEditor.GE_ViewMenu

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----javax.swing.JComponent
                           |
                           +----javax.swing.AbstractButton
                                   |
                                   +----
                                           |
                                           +----
```

javax.swing.JMenuItem

javax.swing.JMenu

caps.GraphEditor.GE_ViewMenu

public class GE_ViewMenu

extends javax.swing.JMenu

implements java.awt.event.ActionListener

Constructs the View menu of the menubar. Also handles the events associated with the View Menu.

Variable Index

● <u>colorMenuItem</u>

    Initiates the 'Color' event

● <u>fontMenuItem</u>

    Initiates the 'Font' event

● <u>manager</u>

● <u>parentFrame</u>

● <u>refreshMenuItem</u>

    Initiates the 'Refresh' event

● <u>selectionModeMenuItem</u>

● <u>toolTipsMenuItem</u>

● <u>treeViewMenuItem</u>

    Initiates the Tree View' event

Constructor Index

GE_ViewMenu(Editor)

The constructor for the View menu

Method Index

•actionPerformed(ActionEvent)

Handles the menu events that occur when one of the menu items is selected

Variables

●**colorMenuItem**

private javax.swing.JMenuItem **colorMenuItem**

Initiates the 'Color' event

●**fontMenuItem**

private javax.swing.JMenuItem **fontMenuItem**

Initiates the 'Font' event

●**refreshMenuItem**

private javax.swing.JMenuItem **refreshMenuItem**

Initiates the 'Refresh' event

●**treeViewMenuItem**

private javax.swing.JCheckBoxMenuItem **treeViewMenuItem**

Initiates the 'Tree View' event

●**toolTipsMenuItem**

private javax.swing.JCheckBoxMenuItem **toolTipsMenuItem**

●**selectionModeMenuItem**

private javax.swing.JCheckBoxMenuItem **selectionModeMenuItem**

●**manager**

private javax.swing.ToolTipManager **manager**

●**parentFrame**

private Editor **parentFrame**

Constructors

●GE_ViewMenu

public **GE_ViewMenu**(Editor parent)

The constructor for the View menu

Methods

●**actionPerformed**

public void **actionPerformed**(java.awt.event.ActionEvent e)

Handles the menu events that occur when one of the menu items is selected

Parameters:

e - The associated ActionEvent

144

Class caps.GraphEditor.IdListEditor

```
java.lang.Object
    |
    +----caps.GraphEditor.IdListEditor
```

public class **IdListEditor**

extends java.lang.Object

implements java.awt.event.ActionListener

Variable Index

• addButton

• cancelButton

• deleteButton

• dialog

• editButton

• HEIGHT

• helpButton

• inputArea

• model

• okButton

• parentFrame

• promptLabel

• south

• vector

Variables

•**dialog**

private static javax.swing.JDialog **dialog**

•**south**

private static javax.swing.JPanel **south**

•**WIDTH**

private static final int **WIDTH**

•**HEIGHT**

private static final int **HEIGHT**

•**vector**

protected static java.util.Vector **vector**

•**okButton**

protected static javax.swing.JButton **okButton**

•**cancelButton**

protected static javax.swing.JButton **cancelButton**

•**helpButton**

protected static javax.swing.JButton **helpButton**

•**addButton**

146

protected static javax.swing.JButton **addButton**

●**deleteButton**

protected static javax.swing.JButton **deleteButton**

●**editButton**

protected static javax.swing.JButton **editButton**

●**inputArea**

protected static javax.swing.JList **inputArea**

●**model**

protected static javax.swing.DefaultListModel **model**

●**promptLabel**

protected static javax.swing.JLabel **promptLabel**

●**parentFrame**

protected Editor **parentFrame**

Constructors

○**IdListEditor**

public **IdListEditor**(Editor parent)

Methods

●**initialize**

protected void **initialize**()

●**openDialog**

public static void **openDialog**(java.util.Vector v)

●**setListElements**

public static void **setListElements**()

●**getIDList**

public static java.util.Vector **getIDList**()

●**actionPerformed**

public void **actionPerformed**(java.awt.event.ActionEvent e)

●**showErrorDialog**

public void **showErrorDialog**(java.lang.String str)

●**showInputDialog**

public java.lang.String **showInputDialog**()

●**showEditDialog**

public java.lang.String **showEditDialog**(java.lang.String initial)

147

Class caps.GraphEditor.Popup

```
java.lang.Object
   |
   +----java.awt.Component
        |
        +----java.awt.Container
             |
             +----javax.swing.JComponent
                  |
                  +----javax.swing.JPopupMenu
                       |
                       +----
```

caps.GraphEditor.Popup

public class **Popup**

extends javax.swing.JPopupMenu

Variable Index

● colorMenuItem

● decomposeMenuItem

● deleteMenuItem

● fontMenuItem

● panel

● propMenuItem

Constructor Index

● Popup(DrawPanel)

Method Index

● getColorMenuItem()

● getDecomposeMenuItem()

● getDeleteMenuItem()

148

Variables

●**decomposeMenuItem**
javax.swing.JMenuItem **decomposeMenuItem**

●**fontMenuItem**
javax.swing.JMenuItem **fontMenuItem**

●**colorMenuItem**
javax.swing.JMenuItem **colorMenuItem**

●**deleteMenuItem**
javax.swing.JMenuItem **deleteMenuItem**

●**propMenuItem**
javax.swing.JMenuItem **propMenuItem**

●**panel**
DrawPanel **panel**

149

Constructors

●**Popup**
public **Popup**(DrawPanel parent)

Methods

●**getDecomposeMenuItem**
public javax.swing.JMenuItem **getDecomposeMenuItem**()

●**getFontMenuItem**
public javax.swing.JMenuItem **getFontMenuItem**()

●**getColorMenuItem**
public javax.swing.JMenuItem **getColorMenuItem**()

●**getDeleteMenuItem**
public javax.swing.JMenuItem **getDeleteMenuItem**()

●**getPropMenuItem**
public javax.swing.JMenuItem **getPropMenuItem**()

●**showPopupMenu**
public void **showPopupMenu**(boolean isEdge,
                          int x,
                          int y)

Class caps.GraphEditor.PrintJob

```
java.lang.Object
   |
   +----caps.GraphEditor.PrintJob
```

public class **PrintJob**

extends java.lang.Object

implements java.lang.Runnable, java.awt.print.Printable,
java.awt.print.Pageable

Constructor Index

● PrintJob(DrawPanel, Vertex)

Method Index

● getNumberOfPages()

● getPageFormat(int)

● getPrintable(int)

● print(DrawPanel, Vertex)

● print(Graphics, PageFormat, int)

● run()

Variable Index

● format

● orientation

● panel

● printablePages

● printJob

150

Variables

●**printablePages**
java.util.Vector **printablePages**

●**printJob**
java.awt.print.PrinterJob **printJob**

●**format**
java.awt.print.PageFormat **format**

●**panel**
DrawPanel **panel**

●**orientation**
int **orientation**

Constructors

●**PrintJob**
public **PrintJob**(DrawPanel p,
                    Vertex root)

Methods

●**run**
public void **run**()

●**print**
public static void **print**(DrawPanel p,
                            Vertex root)

●**print**
public int **print**(java.awt.Graphics g,
                    java.awt.print.PageFormat f,
                    int pi)

●**getNumberOfPages**
public int **getNumberOfPages**()

●**getPageFormat**
public java.awt.print.PageFormat **getPageFormat**(int
pageIndex)

●**getPrintable**
public java.awt.print.Printable **getPrintable**(int pageIndex)

Class caps.GraphEditor.StatusBar

```
java.lang.Object
   |
   +----java.awt.Component
          |
          +----java.awt.Container
                 |
                 +----javax.swing.JComponent
                        |
                        +----javax.swing.JLabel
                               |
                               +----
```

caps.GraphEditor.StatusBar

public class **StatusBar**

extends javax.swing.JLabel

implements java.awt.event.MouseListener

Variable Index

• parent

152

Constructor Index

• StatusBar(Editor)

Method Index

• mouseClicked(MouseEvent)

• mouseDragged(MouseEvent)

• mouseEntered(MouseEvent)

• mouseExited(MouseEvent)

• mousePressed(MouseEvent)

• mouseReleased(MouseEvent)

Variables

• parent

Class caps.GraphEditor.TextEditor

```
java.lang.Object
   |
   +----caps.GraphEditor.TextEditor
```

public class **TextEditor**

extends java.lang.Object

implements java.awt.event.ActionListener

Variable Index

● allowsEmptyString

● cancelButton

● dialog

● grammarKind

Editor **parent**

Constructors

◉ **StatusBar**

public **StatusBar**(Editor e)

Methods

● **mouseEntered**

public void **mouseEntered**(java.awt.event.MouseEvent e)

● **mouseExited**

public void **mouseExited**(java.awt.event.MouseEvent e)

● **mouseClicked**

public void **mouseClicked**(java.awt.event.MouseEvent e)

● **mousePressed**

public void **mousePressed**(java.awt.event.MouseEvent e)

● **mouseDragged**

public void **mouseDragged**(java.awt.event.MouseEvent e)

● **mouseReleased**

public void **mouseReleased**(java.awt.event.MouseEvent e)

153

Constructor Index

• TextEditor(Editor)

Method Index

• actionPerformed(ActionEvent)

• getString()

• initialize()

• openDialog(String, String, String, int, boolean)

• showErrorDialog(String)

Variables

• **dialog**
private static javax.swing.JDialog **dialog**

• **south**
private static javax.swing.JPanel **south**

• HEIGHT

• helpButton

• inputArea

• okButton

• parentFrame

• promptLabel

• south

• text

• WIDTH

WIDTH
private static final int **WIDTH**

HEIGHT
private static final int **HEIGHT**

grammarKind
private static int **grammarKind**

okButton
protected static javax.swing.JButton **okButton**

cancelButton
protected static javax.swing.JButton **cancelButton**

helpButton
protected static javax.swing.JButton **helpButton**

inputArea
protected static javax.swing.JTextArea **inputArea**

promptLabel
protected static javax.swing.JLabel **promptLabel**

allowsEmptyString
static boolean **allowsEmptyString**

text
static java.lang.String **text**

parentFrame
protected Editor **parentFrame**

Constructors

TextEditor

public **TextEditor**(Editor parent)

Methods

initialize
protected void **initialize**()

openDialog
public static void **openDialog**(java.lang.String title,
                                 java.lang.String prompt,
                                 java.lang.String str,
                                 int kind,
                                 boolean flag)

getString
public static java.lang.String **getString**()

actionPerformed
public void **actionPerformed**(java.awt.event.ActionEvent e)

showErrorDialog
public void **showErrorDialog**(java.lang.String str)

155

Class caps.GraphEditor.ToolBar

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----javax.swing.JComponent
                           |
                           +----javax.swing.JToolBar
                                   |
                                   +----
```

caps.GraphEditor.ToolBar

public class **ToolBar**

extends javax.swing.JToolBar

implements java.awt.event.ActionListener

The main toolbar for the prototyping events. Also handles the events associated with the toolbar buttons.

Variable Index

● graphDesc

Initiates the 'Graph Desc' event

● operator

Initiates the 'Operator' event

● parentFrame

the JFrame that is the owner of this toolbar.

● parentSpecs

Initiates the 'Parent Specs' event

● select

Initiates the 'Select' event

● stream

Initiates the 'Stream' event

● terminator

Initiates the 'Terminator' event

● timers

Initiates the 'Timers' event

● types

Initiates the 'Types' event

Constructor Index

● ToolBar(Editor)

156

Constructs a new ToolBar object

Method Index

• actionPerformed(ActionEvent)

Handles the action events that occur when one of the buttons in this toolbar is selected

• enableSelectButton()

This method is called after another operation is finished associated with another button in the toolbar.

• setOperatorButton(boolean)

Variables

● operator

private javax.swing.JButton **operator**

Initiates the 'Operator' event

● terminator

private javax.swing.JButton **terminator**

Initiates the 'Terminator' event

● stream

private javax.swing.JButton **stream**

Initiates the 'Stream' event

● select

private javax.swing.JButton **select**

Initiates the 'Select' event

● types

private javax.swing.JButton **types**

Initiates the 'Types' event

● parentSpecs

private javax.swing.JButton **parentSpecs**

Initiates the 'Parent Specs' event

● timers

private javax.swing.JButton **timers**

Initiates the 'Timers' event

● graphDesc

private javax.swing.JButton **graphDesc**

Initiates the 'Graph Desc' event

● parentFrame

protected Editor **parentFrame**

the JFrame that is the owner of this toolbar.

Constructors

○ ToolBar

public **ToolBar**(Editor frame)

Constructs a new ToolBar object

Parameters:

Class caps.GraphEditor.TreePanel

```
java.lang.Object
    |
    +----java.awt.Component
            |
            +----java.awt.Container
                    |
                    +----javax.swing.JComponent
                            |
                            +----javax.swing.JTree
                                    |
                                    +----
```

caps.GraphEditor.TreePanel

public class **TreePanel**

extends javax.swing.JTree

implements javax.swing.event.TreeSelectionListener,
javax.swing.event.TreeModelListener

The treepanel is the place where the hierarchic structure of the prototype is displayed.

---

frame - The parent frame of this toolbar object.

Methods

●**enableSelectButton**

public void **enableSelectButton**()

This method is called after another operation is finished associated with another button in the toolbar. For example, When an operator is drawn on the DrawPanel, the toolbar will go into select mode.

●**setOperatorButton**

public void **setOperatorButton**(boolean flag)

●**actionPerformed**

public void **actionPerformed**(java.awt.event.ActionEvent e)

Handles the action events that occur when one of the buttons in this toolbar is selected

Parameters:

e - The associated ActionEvent

Variable Index

- **model**

- **parentFrame**

  the JFrame that is the owner of this panel.

Constructor Index

- TreePanel(Editor, Vertex)

  Constructs a new TreePanel object

Method Index

- addNewDFC(DataFlowComponent, DataFlowComponent)

- removeDfc(DataFlowComponent)

- treeNodesChanged(TreeModelEvent)

- treeNodesInserted(TreeModelEvent)

- treeNodesRemoved(TreeModelEvent)

- treeStructureChanged(TreeModelEvent)

- valueChanged(TreeSelectionEvent)

Variables

- **parentFrame**

  Editor **parentFrame**

  the JFrame that is the owner of this panel.

- **model**

  javax.swing.tree.DefaultTreeModel **model**

Constructors

- **TreePanel**

  public **TreePanel** (Editor frame,
  Vertex root)

  Constructs a new TreePanel object

  Parameters:

  frame - The parent frame of this treepanel object.

Class caps.GraphEditor.TreePanelRenderer

```
java.lang.Object
    |
    +----java.awt.Component
            |
            +----java.awt.Container
                    |
                    +----javax.swing.JComponent
                            |
                            +----javax.swing.JLabel
                                    |
                                    +----
```

caps.GraphEditor.TreePanelRenderer

public class **TreePanelRenderer**

extends javax.swing.JLabel

implements javax.swing.tree.TreeCellRenderer

Variable Index

● defaultFont

● opAtomicIcon

Methods

● **addNewDFC**

public void **addNewDFC**(DataFlowComponent dfc,
                    DataFlowComponent parent)

● **removeDfc**

public void **removeDfc**(DataFlowComponent dfc)

● **valueChanged**

public void
valueChanged(javax.swing.event.TreeSelectionEvent e)

● **treeNodesChanged**

public void
treeNodesChanged(javax.swing.event.TreeModelEvent e)

● **treeNodesInserted**

public void
treeNodesInserted(javax.swing.event.TreeModelEvent e)

● **treeNodesRemoved**

public void
treeNodesRemoved(javax.swing.event.TreeModelEvent e)

● **treeStructureChanged**

public void
treeStructureChanged(javax.swing.event.TreeModelEvent e)

160

Method Index

- getTreeCellRendererComponent(JTree, Object, boolean, boolean, int, boolean)

  This is messaged from JTree whenever it needs to get the size of the component or it wants to draw it.

- paint(Graphics)

  paint is subclassed to draw the background correctly.

Variables

- defaultFont

  `protected static java.awt.Font defaultFont`

- termCompositeIcon

  `protected static javax.swing.ImageIcon termCompositeIcon`

- termAtomicIcon

  `protected static javax.swing.ImageIcon termAtomicIcon`

- opCompositeIcon

  `protected static javax.swing.ImageIcon opCompositeIcon`

- opAtomicIcon

  `protected static javax.swing.ImageIcon opAtomicIcon`

- streamIcon

  `protected static javax.swing.ImageIcon streamIcon`

- stateStreamIcon

  `protected static javax.swing.ImageIcon stateStreamIcon`

- opCompositeIcon

- selected

  Whether or not the item that was last configured is selected.

- SelectedBackgroundColor

  Color to use for the background when selected.

- stateStreamIcon

- streamIcon

- termAtomicIcon

- termCompositeIcon

Constructor Index

- TreePanelRenderer()

●**SelectedBackgroundColor**

```
protected static final java.awt.Color
SelectedBackgroundColor
```

    Color to use for the background when selected.

●**selected**

```
protected boolean selected
```

    Whether or not the item that was last configured is selected.

Constructors

◗**TreePanelRenderer**

```
public TreePanelRenderer()
```

Methods

●**getTreeCellRendererComponent**

```
public java.awt.Component
getTreeCellRendererComponent(javax.swing.JTree tree,
java.lang.Object value,
                                                    boolean
selected,
                                                    boolean
expanded,
                                                    boolean leaf,
                                                    int row,
                                                    boolean
hasFocus)
```

    This is messaged from JTree whenever it needs to get the size of the component or it wants to draw it. This attempts to set the font based on value, which will be a TreeNode.

●**paint**

```
public void paint(java.awt.Graphics g)
```

    paint is subclassed to draw the background correctly. JLabel currently does not allow backgrounds other than white, and it will also fill behind the icon. Something that isn't desirable.

Overrides:

    <u>paint</u> in class javax.swing.JComponent

162

Class caps.GraphEditor.VertexProperties

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Dialog
                                   |
                                   +----
```
```
javax.swing.JDialog
                                           |
                                           +----
```
```
caps.GraphEditor.VertexProperties
```

public class **VertexProperties**

extends javax.swing.JDialog

implements java.awt.event.ActionListener

Variable Index

• cancelButton

• changeStatus

• dVertex

• exceptionGuardsButton

• exceptionListButton

• finishWithinLabel

• formalDescButton

• fwField

• fwReqByButton

• fwUnitsCombo

• guardsPanel

- helpButton
- ifCondField
- ifConditionButton
- informalDescButton
- keywordsButton
- keywordsPanel
- languageCombo
- metField
- metLabel
- metReqByButton

- metUnitsCombo
- nameField
- namePanel
- okButton
- okPanel
- operatorCombo
- outputGuardsButton
- parentFrame
- periodField
- periodLabel

164

- periodReqByButton
- periodUnitsCombo
- targetVertex
- tempVertex
- timerOpsButton
- timingCombo
- timingPanel
- TO_OPERATOR
- TO_TERMINATOR
- triggerCombo
- triggerPanel
- triggerReqByButton
- triggerStreamsButton
- UNCHANGED

Constructor Index

- VertexProperties(Editor)

Method Index

- actionPerformed(ActionEvent)
- getUnitsCombo()
- initialize()

- resetTimingPanelComponents()

- setButtonText(JButton, Object)

- setDisplayVertex(DisplayVertex)

- setVertex(Vertex)

- showErrorDialog(String)

Variables

- **TO_OPERATOR**
public static final int **TO_OPERATOR**

- **TO_TERMINATOR**
public static final int **TO_TERMINATOR**

- **UNCHANGED**
public static final int **UNCHANGED**

- **changeStatus**
private int **changeStatus**

- **targetVertex**
Vertex **targetVertex**

- **dVertex**
DisplayVertex **dVertex**

- **namePanel**
javax.swing.JPanel **namePanel**

- **triggerPanel**
javax.swing.JPanel **triggerPanel**

- **timingPanel**
javax.swing.JPanel **timingPanel**

- **guardsPanel**
javax.swing.JPanel **guardsPanel**

- **keywordsPanel**
javax.swing.JPanel **keywordsPanel**

- **okPanel**
javax.swing.JPanel **okPanel**

- **nameField**
javax.swing.JTextField **nameField**

- **ifCondField**
java.awt.TextArea **ifCondField**

- **metField**
javax.swing.JTextField **metField**

- **periodField**
javax.swing.JTextField **periodField**

- **fwField**
javax.swing.JTextField **fwField**

- **metLabel**
javax.swing.JLabel **metLabel**

- **periodLabel**
  javax.swing.JLabel **periodLabel**
- **finishWithinLabel**
  javax.swing.JLabel **finishWithinLabel**
- **operatorCombo**
  javax.swing.JComboBox **operatorCombo**
- **languageCombo**
  javax.swing.JComboBox **languageCombo**
- **triggerCombo**
  javax.swing.JComboBox **triggerCombo**
- **timingCombo**
  javax.swing.JComboBox **timingCombo**
- **metUnitsCombo**
  javax.swing.JComboBox **metUnitsCombo**
- **periodUnitsCombo**
  javax.swing.JComboBox **periodUnitsCombo**
- **fwUnitsCombo**
  javax.swing.JComboBox **fwUnitsCombo**
- **ifConditionButton**
  javax.swing.JButton **ifConditionButton**
- **triggerReqByButton**
  javax.swing.JButton **triggerReqByButton**
- **metReqByButton**
  javax.swing.JButton **metReqByButton**
- **periodReqByButton**
  javax.swing.JButton **periodReqByButton**

- **fwReqByButton**
  javax.swing.JButton **fwReqByButton**
- **outputGuardsButton**
  javax.swing.JButton **outputGuardsButton**
- **exceptionGuardsButton**
  javax.swing.JButton **exceptionGuardsButton**
- **exceptionListButton**
  javax.swing.JButton **exceptionListButton**
- **timerOpsButton**
  javax.swing.JButton **timerOpsButton**
- **keywordsButton**
  javax.swing.JButton **keywordsButton**
- **informalDescButton**
  javax.swing.JButton **informalDescButton**
- **formalDescButton**
  javax.swing.JButton **formalDescButton**
- **okButton**
  javax.swing.JButton **okButton**
- **cancelButton**
  javax.swing.JButton **cancelButton**
- **helpButton**
  javax.swing.JButton **helpButton**
- **triggerStreamsButton**
  javax.swing.JButton **triggerStreamsButton**
- **parentFrame**
  Editor **parentFrame**

●**tempVertex**
Vertex **tempVertex**

Constructors

◎**VertexProperties**
public **VertexProperties**(Editor parent)

Methods

●**initialize**
public void **initialize**()
●**getUnitsCombo**
public javax.swing.JComboBox **getUnitsCombo**()
●**setVertex**
public void **setVertex**(Vertex v)
●**setDisplayVertex**
public void **setDisplayVertex**(DisplayVertex v)
●**resetTimingPanelComponents**
public void **resetTimingPanelComponents**()
●**actionPerformed**
public void **actionPerformed**(java.awt.event.ActionEvent e)
●**showErrorDialog**
public void **showErrorDialog**(java.lang.String str)
●**setButtonText**

public void **setButtonText**(javax.swing.JButton b,
                             java.lang.Object o)

Class caps.Psdl.DataFlowComponent

```
java.lang.Object
   |
   +----javax.swing.tree.DefaultMutableTreeNode
        |
        +----caps.Psdl.DataFlowComponent
```

public abstract class **DataFlowComponent**

extends javax.swing.tree.DefaultMutableTreeNode

DataFlowComponent is the abstract base class of the Vertex and Edge classes. It extends DefaultMutableTreeNode, so every object of this class is actually a tree node.

Variable Index

• id

The id of this component

• label

The label to display on the DrawPanel

• labelFont

The font parameter of the label.

• labelXOffset

The x-offset of the label from the center of the component

• labelYOffset

The y-offset of the label from the center of the component

• lFont

The font representation of the label.

• met

The met of a Vertex or the latency of a Stream.

• metFont

The font parameter of the met label of this component.

• metlFont

The font representation of the met (or latency).

• metXOffset

The x-offset of the met label from the center of this component.

• metYOffset

The y-offset of the met label from the center of this component.

• UNIQUE_ID

The unique id of components.

169

Constructor Index

● DataFlowComponent(Vertex)

 The constructor for this class.

Method Index

● getId()

 Returns the id of this component.

● getLabel()

 Returns the label of this component.

● getLabelFontIndex()

 Returns the label font index of this component.

● getLabelXOffset()

 Returns the x-component of the offset of the label.

● getLabelYOffset()

 Returns the y-component of the offset of the label.

● getlFont()

 Returns font of the label.

● getMet()

 Returns the met (or latency) of this component.

● getMetFontIndex()

 Returns the met (or latency) font index of this component.

● getMetlFont()

 Returns font of the met (or latency).

● getMetXOffset()

 Returns the x-component of the offset of the met (or latency).

● getMetYOffset()

 Returns the y-component of the offset of the met (or latency).

● getX()

 This abstract method is implemented in the subclasses.

● getY()

 This abstract method is implemented in the subclasses.

● moveTo(int, int)

 This abstract method is implemented in the subclasses.

● setId(int)

 Sets the id of this component to the specified value.

● setLabel(String)

 Sets the label of this component to the specified value.

● setLabelFontIndex(int)

 Changes the label font index to the specified value.

Variables

●**label**

protected java.lang.String **label**

    The label to display on the DrawPanel

●**UNIQUE_ID**

protected static int **UNIQUE_ID**

    The unique id of components.

●**id**

protected int **id**

    The id of this component

●**labelFont**

protected int **labelFont**

    The font parameter of the label.

●**lFont**

protected java.awt.Font **lFont**

    The font representation of the label.

●**labelXOffset**

protected int **labelXOffset**

    The x-offset of the label from the center of the component

●**labelYOffset**

protected int **labelYOffset**

    The y-offset of the label from the center of the component

●**met**

●setLabelOffset(int, int)

    Changes the label offset to the specified x and y values.

●setLabelXOffset(int)

    Sets the x-component of the offset of the label to the specified value.

●setLabelYOffset(int)

    Sets the y-component of the offset of the label to the specified value.

●setMet(PSDLTime)

    Sets the met (or latency) of this component to the specified value.

●setMetFontIndex(int)

    Changes the met (or latency) font index to the specified value.

●setMetOffset(int, int)

    Changes the met (or latency) offset to the specified x and y values.

●setMetXOffset(int)

    Sets the x-component of the offset of the met (or latency) to the specified value.

●setMetYOffset(int)

    Sets the y-component of the offset of the met (or latency) to the specified value.

●toString()

    Returns the name (label) of this component.

protected PSDLTime **met**

    The met of a Vertex or the latency of a Stream.

● **metFont**

protected int **metFont**

    The font parameter of the met label of this component.

● **metXOffset**

protected int **metXOffset**

    The x-offset of the met label from the center of this component.

● **metYOffset**

protected int **metYOffset**

    The y-offset of the met label from the center of this component.

● **metlFont**

protected java.awt.Font **metlFont**

    The font representation of the met (or latency).

Constructors

◑ **DataFlowComponent**

protected **DataFlowComponent** (Vertex v)

    The constructor for this class.

Parameters:

    v - The parent vertex of this component

172

Methods

● **getId**

public int **getId** ()

    Returns the id of this component.

● **setId**

public void **setId** (int i)

    Sets the id of this component to the specified value.

● **setLabel**

public void **setLabel** (java.lang.String s)

    Sets the label of this component to the specified value.

● **getLabel**

public java.lang.String **getLabel** ()

    Returns the label of this component.

● **getLabelXOffset**

public int **getLabelXOffset** ()

    Returns the x-component of the offset of the label.

● **setLabelXOffset**

public void **setLabelXOffset** (int xLoc)

    Sets the x-component of the offset of the label to the specified value.

● **setLabelYOffset**

public void **setLabelYOffset** (int yLoc)

    Sets the y-component of the offset of the label to the specified

● **setMetYOffset**

public void **setMetYOffset**(int yLoc)

Sets the y-component of the offset of the met (or latency) to the specified value.

● **getMetYOffset**

public int **getMetYOffset**()

Returns the y-component of the offset of the met (or latency).

● **setMetOffset**

public void **setMetOffset**(int xOffset,
                            int yOffset)

Changes the met (or latency) offset to the specified x and y values.

● **getMetlFont**

public java.awt.Font **getMetlFont**()

Returns font of the met (or latency).

● **getX**

public abstract int **getX**()

This abstract method is implemented in the subclasses.

● **getY**

public abstract int **getY**()

This abstract method is implemented in the subclasses.

● **moveTo**

public abstract void **moveTo**(int xOffset,
                               int yOffset)

This abstract method is implemented in the subclasses.

● **toString**

value.

● **getLabelYOffset**

public int **getLabelYOffset**()

Returns the y-component of the offset of the label.

● **setLabelOffset**

public void **setLabelOffset**(int xOffset,
                              int yOffset)

Changes the label offset to the specified x and y values.

● **getlFont**

public java.awt.Font **getlFont**()

Returns font of the label.

● **setMet**

public void **setMet**(PSDLTime s)

Sets the met (or latency) of this component to the specified value.

● **getMet**

public PSDLTime **getMet**()

Returns the met (or latency) of this component.

● **getMetXOffset**

public int **getMetXOffset**()

Returns the x-component of the offset of the met (or latency).

● **setMetXOffset**

public void **setMetXOffset**(int xLoc)

Sets the x-component of the offset of the met (or latency) to the specified value.

Class caps.Psdl.DataTypes

```
java.lang.Object
   |
   +----caps.Psdl.DataTypes
```

public class **DataTypes**

extends java.lang.Object

Variable Index

● impls

● specs

● types

---

public java.lang.String **toString** ()

Returns the name (label) of this component.

Overrides:

toString in class javax.swing.tree.DefaultMutableTreeNode

● **setLabelFontIndex**

public void **setLabelFontIndex** (int f)

Changes the label font index to the specified value.

● **getLabelFontIndex**

public int **getLabelFontIndex** ()

Returns the label font index of this component.

● **setMetFontIndex**

public void **setMetFontIndex** (int f)

Changes the met (or latency) font index to the specified value.

● **getMetFontIndex**

public int **getMetFontIndex** ()

Returns the met (or latency) font index of this component.

174

Constructor Index

● DataTypes()

Method Index

● addType(String)

● addType(String, String, String)

● build(StreamTokenizer)

● buildTypes(File)

● buildTypes(String)

● exists(String)

● getNextToken(StreamTokenizer)

● isPredefined(String)

● removeElements()

● toString()

Variables

● types
private java.util.Vector **types**

● specs
private java.util.Vector **specs**

● impls
private java.util.Vector **impls**

Constructors

● DataTypes
public **DataTypes** ()

Methods

● addType

Class caps.Psdl.Edge

```
java.lang.Object
   |
   +----javax.swing.tree.DefaultMutableTreeNode
           |
           +----caps.Psdl.DataFlowComponent
                   |
                   +----caps.Psdl.Edge
```

public class **Edge**

extends DataFlowComponent

Edge represents a stream in the data flow diagram It is also a TreeNode
object

Variable Index

● destination

The destination Vertex of this stream.

● initialValue

The initial value of the stream.

---

public void **addType**(java.lang.String name,
                       java.lang.String spec,
                       java.lang.String impl)

● **addType**

public void **addType**(java.lang.String name)

● **exists**

public boolean **exists**(java.lang.String name)

● **isPredefined**

public boolean **isPredefined**(java.lang.String str)

● **buildTypes**

public void **buildTypes**(java.io.File file)

● **buildTypes**

public void **buildTypes**(java.lang.String s)

● **build**

private void **build**(java.io.StreamTokenizer tok)

● **getNextToken**

public java.lang.String
getNextToken(java.io.StreamTokenizer tok) throws
java.io.IOException

● **removeElements**

public void **removeElements**()

● **toString**

public java.lang.String **toString**()

Overrides:

toString in class java.lang.Object

Method Index

- **addPoint(int, int)**
  Adds a new point to the control points.
- **correctEndingPoints()**
  Locates the ending points of this stream on the perimeter of the source and destination.
- **correctLabelOffset()**
  Sets the location of this stream to the middle control point.
- **delete()**
  Deletes this stream.
- **delete(boolean)**
  Deletes this stream.
- **deleteHelper()**
  Helper method to delete the stream.
- **getDestination()**
  Returns the destination Vertex of this stream.
- **getInitialValue()**
  Returns the initial value of this stream.
- **getPoints()**

- **isState**
  True if this is a state stream.
- **points**
  The vector that holds the control points of this stream.
- **selectedHandleIndex**
  The index of the handle that the mouse is pressed on.
- **source**
  The source Vertex of this stream.
- **streamType**
  The type name of the stream.
- **x**
  The x location of this stream in the DrawPanel.
- **y**
  The y location of this stream in the DrawPanel.

Constructor Index

- **Edge(int, int, Vertex)**
  The constructor for this class.

177

Returns the control points vector.

• getSource()

Returns the source Vertex of this stream.

• getStreamType()

Returns the type of this stream.

• getX()

Returns the x value of this stream.

• getY()

Returns the y value of this stream.

• isStateStream()

Returns true if this is a state stream.

• moveTo(int, int)

Relocates the stream when the stream is moved with other objects.

• reShape(int, int)

Is called when one of the handles of the stream is dragged in the DrawPanel.

• setDestination(Vertex)

Sets the destination Vertex of this stream to the specified value.

• setInitialControlPoints(String)

Called to extract a string representation of the control points.

• setInitialValue(String)

Sets the initial value of this stream to the specified value.

• setSelectedHandleIndex(int)

Changes selectedHandleIndex to the specified value.

• setSource(Vertex)

Sets the source Vertex of this stream to the specified value.

• setStateStream(boolean)

Changes the isState field to the specified value.

• setStreamType(String)

Sets the type of this stream to the specified value.

• setX(int)

Changes the x value of the stream to the specified value.

• setY(int)

Changes the y value of the stream to the specified value.

Variables

● source

protected Vertex source

The source Vertex of this stream.

● destination

protected Vertex destination

178

The destination Vertex of this stream.

●points

`protected java.util.Vector points`

The vector that holds the control points of this stream.

●streamType

`protected java.lang.String streamType`

The type name of the stream.

●initialValue

`protected java.lang.String initialValue`

The initial value of the stream.

●isState

`protected boolean isState`

True if this is a state stream.

●x

`protected int x`

The x location of this stream in the DrawPanel.

●y

`protected int y`

The y location of this stream in the DrawPanel.

●selectedHandleIndex

`protected int selectedHandleIndex`

The index of the handle that the mouse is pressed on.

179

Constructors

●Edge

```
public Edge(int xLocation,
            int yLocation,
            Vertex v)
```

The constructor for this class.

Parameters:

v - the parent vertex of this edge.

Methods

●moveTo

```
public void moveTo(int xOffset,
                   int yOffset)
```

Relocates the stream when the stream is moved with other objects.

Overrides:

moveTo in class DataFlowComponent

●reShape

```
public void reShape(int xLocation,
                    int yLocation)
```

Is called when one of the handles of the stream is dragged in the DrawPanel.

●setX

`public void setX(int newX)`

Changes the x value of the stream to the specified value.

●**setY**

public void **setY**(int newY)

Changes the y value of the stream to the specified value.

●**setSelectedHandleIndex**

public void **setSelectedHandleIndex**(int i)

Changes selectedHandleIndex to the specified value.

●**getX**

public int **getX**()

Returns the x value of this stream.

Overrides:

getX in class DataFlowComponent

●**getY**

public int **getY**()

Returns the y value of this stream.

Overrides:

getY in class DataFlowComponent

●**getSource**

public Vertex **getSource**()

Returns the source Vertex of this stream.

●**setSource**

public void **setSource**(Vertex v)

Sets the source Vertex of this stream to the specified value.

180

●**getDestination**

public Vertex **getDestination**()

Returns the destination Vertex of this stream.

●**setDestination**

public void **setDestination**(Vertex v)

Sets the destination Vertex of this stream to the specified value.

●**getStreamType**

public java.lang.String **getStreamType**()

Returns the type of this stream.

●**setStreamType**

public void **setStreamType**(java.lang.String type)

Sets the type of this stream to the specified value.

●**isStateStream**

public boolean **isStateStream**()

Returns true if this is a state stream.

●**setStateStream**

public void **setStateStream**(boolean flag)

Changes the isState field to the specified value.

●**getInitialValue**

public java.lang.String **getInitialValue**()

Returns the initial value of this stream.

●**setInitialValue**

public void **setInitialValue**(java.lang.String str)

Sets the initial value of this stream to the specified value.

addPoint

```
public void addPoint(int x,
                     int y)
```

Adds a new point to the control points. Also adds the middle point of the control points.

Parameters:

x - the x component of the new conrol point.

y - the y component of the new conrol point.

getPoints

```
public java.util.Vector getPoints()
```

Returns the control points vector.

correctLabelOffset

```
public void correctLabelOffset()
```

Sets the location of this stream to the middle control point.

correctEndingPoints

```
public void correctEndingPoints()
```

Locates the ending points of this stream on the perimeter of the source and destination.

setInitialControlPoints

```
public void setInitialControlPoints(java.lang.String exp)
```

Called to extract a string representation of the control points. Constructs the points vector from the string expression.

delete

```
public void delete(boolean deletingInEdge)
```

Deletes this stream.

delete

```
public void delete()
```

Deletes this stream.

deleteHelper

```
public void deleteHelper()
```

Helper method to delete the stream.

Class caps.Psdl.External

```
java.lang.Object
   |
   +----javax.swing.tree.DefaultMutableTreeNode
           |
           +----caps.Psdl.DataFlowComponent
                   |
                   +----caps.Psdl.Vertex
                           |
                           +----caps.Psdl.External
```

public class **External**

extends Vertex

Constructors

○External

```
public External(int xLocation,
                int yLocation,
                Vertex v)
```

Methods

●getIntersectionPoint

```
public java.awt.Point getIntersectionPoint(java.awt.Point
p)
```

Overrides:

getIntersectionPoint in class Vertex

Constructor Index

○External(int, int, Vertex)

Method Index

●getIntersectionPoint(Point)

182

Class caps.Psdl.PSDLTime

```
java.lang.Object
   |
   +----caps.Psdl.PSDLTime
```

public class **PSDLTime**

extends java.lang.Object

This class represents a combination of time value from an integer that represents the time and another integer that represents the unit.

Variable Index

● **hours**

The constant value for hours.

● **microsec**

The constant value for microseconds.

● **min**

The constant value for minutes.

● **ms**

The constant value for miliseconds.

● **sec**

The constant value for seconds.

● **units**

The units of the time.

● **value**

The value of the time.

Constructor Index

● **PSDLTime()**

The constructor for this class.

● **PSDLTime(int, int)**

The constructor for this class.

Method Index

● **getTimeUnits()**

Returns the time units of this object.

● **getTimeValue()**

Returns the time time value of this object.

183

- setTimeUnits(int)

  Sets the time unit to the specified argument.

- setTimeUnits(String)

  Sets the time unit to the specified argument.

- setTimeValue(int)

  Sets the time value to the specified argument.

- toString()

  Returns a string representation of this object.

Variables

- **microsec**

  `public static final int microsec`

  The constant value for microseconds.

- **ms**

  `public static final int ms`

  The constant value for miliseconds.

- **sec**

  `public static final int sec`

  The constant value for seconds.

- **min**

  `public static final int min`

  The constant value for minutes.

- **hours**

  `public static final int hours`

  The constant value for hours.

- **value**

  `private int value`

  The value of the time.

- **units**

  `private int units`

  The units of the time.

Constructors

- **PSDLTime**

  `public PSDLTime()`

  The constructor for this class.

- **PSDLTime**

  `public PSDLTime(int timeValue, int timeUnits)`

  The constructor for this class.

  Parameters:

  timeValue - the value of the time.

  timeUnits - the unit of the time.

Methods

● **getTimeValue**

public int **getTimeValue**()

Returns the time time value of this object.

● **setTimeValue**

public void **setTimeValue**(int timeValue)

Sets the time value to the specified argument.

● **getTimeUnits**

public int **getTimeUnits**()

Returns the time units of this object.

● **setTimeUnits**

public void **setTimeUnits**(int timeUnits)

Sets the time unit to the specified argument.

● **setTimeUnits**

public void **setTimeUnits**(java.lang.String u)

Sets the time unit to the specified argument.

● **toString**

public java.lang.String **toString**()

Returns a string representation of this object.

Returns:

the string representation in the form of "12 sec"

Overrides:

185

Class caps.Psdl.Vertex

```
java.lang.Object
    |
    +----javax.swing.tree.DefaultMutableTreeNode
            |
            +----caps.Psdl.DataFlowComponent
                    |
                    +----caps.Psdl.Vertex
```
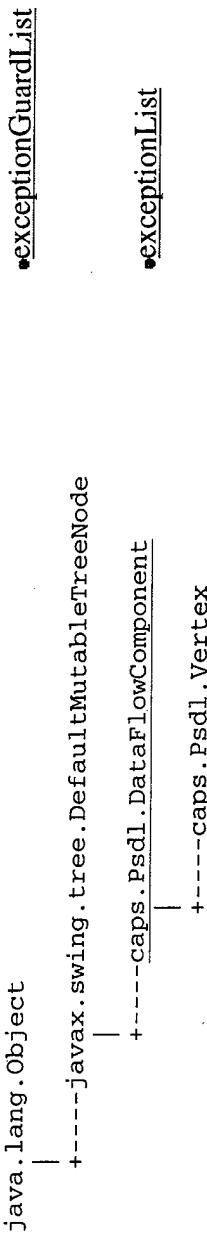
public class **Vertex**

extends DataFlowComponent

This class represents a terminator or an operator. It holds the data structures that represent the constructs for the Vertex.

Variable Index

● **BY_ALL**

The constant value for Vertices that have "BY ALL" triggering construct.

● **BY_SOME**

The constant value for Vertices that have "BY SOME" triggering construct.

● **color**

The color parameter of this component.

● **exceptionGuardList**

● **exceptionList**

● **finishWithin**

● **finishWithinReqmts**

● **formalDesc**

● **genericList**

● **graphDesc**

● **height**

The height of this component.

● **ifCondition**

- impLanguage

- inEdges

- informalDesc

- INITIAL_RADIUS

  The constant value for the intitial radius of a Vertex.

- keywordList

- mcp

- mcpReqmts

- metReqmts

- mrt

- mrtReqmts

- NON_TIME_CRITICAL

  The constant value for non-time ciritical Vertices.

- outEdges

- outputGuardList

- period

- PERIODIC

  The constant value for periodic Vertices.

- periodReqmts

- specReqmts

- SPORADIC

  The constant value for sporadic Vertices.

- terminator

  True if this Vertex is a terminator.

- timerList

Constructor Index

● Vertex(int, int, Vertex, boolean)

　　The constructor for this class.

Method Index

● addInEdge(Edge)

　　Adds a new Edge to the inEdges Vector.

● addOutEdge(Edge)

　　Adds a new Edge to the outEdges Vector.

● correctInOutStreams()

　　Corrects the ending points of the in and out streams of this component.

● delete()

　　Deletes this Vertex.

● extractList(Vector)

　　Extracts an idList which is represented as a Vector and returns a String representation of the idList so that it will have the form "id1, id2, id3..."

● extractString(String, boolean)

　　Called from getSpecification.

● getColor()

● timerOpList

● timingType

● triggerReqmts

● triggerStreamsList

● triggerType

● UNPROTECTED

　　The constant value for unprotected Vertices.

● width

　　The width of this component.

● x

　　The x-location of this component on the DrawPanel

● y

　　The y-location of this component on the DrawPanel

188

Returns the color value for this Vertex.

- **getExceptionGuardList()**

Returns the exception guard list of this Vertex.

- **getExceptionList()**

Returns the exception list of this Vertex.

- **getFinishWithin()**

Returns the finish within value of this Vertex.

- **getFinishWithinReqmts()**

Returns the finish within requirements of this Vertex.

- **getFormalDesc()**

Returns the formal description of this Vertex.

- **getGenericList()**

Returns the generic list of this Vertex.

- **getGraphDesc()**

Returns the informal graph description of this Vertex.

- **getHeight()**

Returns the height of this component.

- **getIfCondition()**

Returns the if condition of this Vertex.

- **getImpLanguage()**

Returns the implementation language of this Vertex.

- **getInformalDesc()**

Returns the informal description of this Vertex.

- **getIntersectionPoint(Point)**

Returns intersection point of this vertex with the specified point.

- **getKeywordList()**

Returns the keywords of this Vertex.

- **getMcp()**

Returns the mcp value of this Vertex.

- **getMcpReqmts()**

Returns the mcp requirements of this Vertex.

- **getMetReqmts()**

Returns the met requirements of this Vertex.

- **getMrt()**

Returns the mrt value of this Vertex.

- **getMrtReqmts()**

Returns the mrt requirements of this Vertex.

- **getOperatorIntersection(Point)**

Returns the intersection point of this vertex with the specified point.

- **getOutputGuardList()**

Returns the output guard list of this Vertex.

- getPeriod()

Returns the period value of this Vertex.

- getPeriodReqmts()

Returns the period requirements of this Vertex.

- getSpecification(boolean)

Creates the specification construct from its data structures.

- getSpecReqmts()

Returns the spec requirements of this Vertex.

- getTerminatorIntersection(Point)

Returns the intersection point of this vertex with the specified point.

- getTimerList()

Returns the timers of this Vertex.

- getTimerOpList()

Returns the timer op list of this Vertex.

- getTimingType()

Returns the timing type of this Vertex.

- getTriggerReqmts()

Returns the trigger requirements of this Vertex.

- getTriggerStreamsList()

Returns the triggering streams of this Vertex.

- getTriggerType()

Returns the triggering type of this Vertex.

- getWidth()

Returns the width of this component.

- getX()

Returns the x component of the location of this Vertex

- getY()

Returns the y component of the location of this Vertex.

- isTerminator()

Returns true if this component is a terminator.

- moveTo(int, int)

Sets the location of this component on the screen.

- removeInEdge(Edge)

Removes an Edge from the inEdges Vector.

- removeOutEdge(Edge)

Removes an Edge from the outEdges Vector.

- setColor(int)

Changes the color value for this Vertex.

- setExceptionGuardList(String)

Sets the exception guards list to the specified value.

•setExceptionList(String)

Sets the exception list to the specified value.

•setFinishWithin(PSDLTime)

Sets the finish within to the specified value.

•setFinishWithinReqmts(Vector)

Sets the finish within requirements to the specified value.

•setFormalDesc(String)

Sets the formal description to the specified value.

•setGenericList(String)

Sets the generic list to the specified value.

•setGraphDesc(String)

Sets the graph description to the specified value.

•setIfCondition(String)

Sets the if condition to the specified value.

•setImpLanguage(String)

Sets the implementation language to the specified value.

•setInformalDesc(String)

Sets the informal description to the specified value.

•setKeywordList(Vector)

Sets the keywords to the specified value.

•setLocation(int, int)

Sets the location of this component on the screen.

•setMcp(PSDLTime)

Sets the mcp to the specified value.

•setMcpReqmts(Vector)

Sets the mcp requirements to the specified value.

•setMetReqmts(Vector)

Sets the met requirements to the specified value.

•setMrt(PSDLTime)

Sets the mrt to the specified value.

•setMrtReqmts(Vector)

Sets the mrt requirements to the specified value.

•setOutputGuardList(String)

Sets the output guard list to the specified value.

•setPeriod(PSDLTime)

Sets the period to the specified value.

•setPeriodReqmts(Vector)

Sets the period reqirements to the specified value.

•setTerminator(boolean)

Variables

●INITIAL_RADIUS

public static final int **INITIAL_RADIUS**

The constant value for the intitial radius of a Vertex.

●NON_TIME_CRITICAL

public static final int **NON_TIME_CRITICAL**

The constant value for non-time ciritical Vertices.

●PERIODIC

public static final int **PERIODIC**

The constant value for periodic Vertices.

●SPORADIC

public static final int **SPORADIC**

The constant value for sporadic Vertices.

●UNPROTECTED

public static final int **UNPROTECTED**

The constant value for unprotected Vertices.

●BY_SOME

public static final int **BY_SOME**

The constant value for Vertices that have "BY SOME" triggering construct.

●BY_ALL

public static final int **BY_ALL**

The constant value for Vertices that have "BY ALL" triggering

Sets this component as a terminator or a stream.

●setTimerList(Vector)

Sets the timer list to the specifed value.

●setTimerOpList(String)

Sets the timer op list to the specified value.

●setTimingType(int)

Sets the timing type to the specified value.

●setTriggerReqmts(Vector)

Sets the trigger requirements to the specified value.

●setTriggerStreamsList(Vector)

Sets the trigger streams list to the specified value.

●setTriggerType(int)

Sets the triggering type to the specified value.

●setWidth(int)

Changes the width of this component.

●setX(int)

Changes the x component of the location of this Vertex

●setY(int)

Changes the y component of the location of this Vertex.

construct.

●terminator

protected boolean **terminator**

True if this Vertex is a terminator.

●x

protected int **x**

The x-location of this component on the DrawPanel

●y

protected int **y**

The y-location of this component on the DrawPanel

●width

protected int **width**

The width of this component. It serves as the radius of an operator and the width of a terminator width of operator .cap

●height

protected int **height**

The height of this component.

●color

protected int **color**

The color parameter of this component.

●metReqmts

protected java.util.Vector **metReqmts**

●period

protected PSDLTime **period**

●periodReqmts

protected java.util.Vector **periodReqmts**

●finishWithin

protected PSDLTime **finishwithin**

●finishWithinReqmts

protected java.util.Vector **finishwithinReqmts**

●mcp

protected PSDLTime **mcp**

●mcpReqmts

protected java.util.Vector **mcpReqmts**

●mrt

protected PSDLTime **mrt**

●mrtReqmts

protected java.util.Vector **mrtReqmts**

●timingType

protected int **timingType**

●triggerType

protected int **triggerType**

●triggerReqmts

protected java.util.Vector **triggerReqmts**

●triggerStreamsList

protected java.util.Vector **triggerStreamsList**

●ifCondition

protected java.lang.String **ifCondition**

●outputGuardList

protected java.lang.String **outputGuardList**

exceptionGuardList

protected java.lang.String **exceptionGuardList**

exceptionList

protected java.lang.String **exceptionList**

timerOpList

protected java.lang.String **timerOpList**

keywordList

protected java.util.Vector **keywordList**

informalDesc

protected java.lang.String **informalDesc**

formalDesc

protected java.lang.String **formalDesc**

inEdges

protected java.util.Vector **inEdges**

outEdges

protected java.util.Vector **outEdges**

impLanguage

protected java.lang.String **impLanguage**

timerList

protected java.util.Vector **timerList**

graphDesc

protected java.lang.String **graphDesc**

genericList

protected java.lang.String **genericList**

specReqmts

protected java.util.Vector **specReqmts**

194

Constructors

◯**Vertex**

public **Vertex**(int xLocation,
                int yLocation,
                <u>Vertex v,</u>
                <u>boolean t)</u>

The constructor for this class.

Parameters:

xLocation - The x component of the location of this component.

yLocation - The y component of the location of this component.

v - The parent vertex of this component.

t - true if this component is a terminator.

Methods

●**setLocation**

public void **setLocation**(int xOffset,
                        int yOffset)

Sets the location of this component on the screen. Also corrects the location of the ending streams.

Parameters:

xLocation - The new x component of the location on the drawpanel

yLocation - The new y component of the location on the drawpanel

●**moveTo**

```
public void moveTo(int xOffset,
                   int yOffset)
```

Sets the location of this component on the screen.

Parameters:

xLocation - The new x component of the location on the drawpanel

yLocation - The new y component of the location on the drawpanel

Overrides:

moveTo in class DataFlowComponent

●**isTerminator**

```
public boolean isTerminator()
```

Returns true if this component is a terminator.

Returns:

true if this component is a terminator.

●**setTerminator**

```
public void setTerminator(boolean b)
```

Sets this component as a terminator or a stream. Also changes the width of the component.

Parameters:

b. -

●**correctInOutStreams**

```
public void correctInOutStreams()
```

Corrects the ending points of the in and out streams of this component.

●**getWidth**

```
public int getWidth()
```

Returns the width of this component.

Returns:

the width of this component.

●**setWidth**

```
public void setWidth(int w)
```

Changes the width of this component.

Parameters:

w - the new width of this component.

●**getHeight**

```
public int getHeight()
```

Returns the height of this component.

Returns:

the height of this component.

●**getX**

```
public int getX()
```

Returns the x component of the location of this Vertex

Returns:

x

Overrides:

getX in class DataFlowComponent

●setX

`public void setX(int xLoc)`

Changes the x component of the location of this Vertex

Parameters:

xLoc. -

●setY

`public void setY(int yLoc)`

Changes the y component of the location of this Vertex.

Parameters:

yLoc. -

●getY

`public int getY()`

Returns the y component of the location of this Vertex.

Returns:

y

Overrides:

getY in class DataFlowComponent

●setColor

`public void setColor(int c)`

Changes the color value for this Vertex.

Parameters:

c - the new color value.

●getColor

`public int getColor()`

Returns the color value for this Vertex.

Returns:

the color value of the Vertex.

●addInEdge

`public void addInEdge(Edge e)`

Adds a new Edge to the inEdges Vector.

Parameters:

e - the new inEdge.

●removeInEdge

`public void removeInEdge(Edge e)`

Removes an Edge from the inEdges Vector.

Parameters:

e - the inEdge to be removed.

●addOutEdge

`public void addOutEdge(Edge e)`

Adds a new Edge to the outEdges Vector.

Parameters:

e - the new outEdge.

●removeOutEdge

`public void removeOutEdge(Edge e)`

Removes an Edge from the outEdges Vector.

Parameters:

e - the outEdge to be removed.

●getTimingType

```
public int getTimingType()
```

Returns the timing type of this Vertex.

●setTimingType

```
public void setTimingType(int type)
```

Sets the timing type to the specified value.

●getTriggerType

```
public int getTriggerType()
```

Returns the triggering type of this Vertex.

●setTriggerType

```
public void setTriggerType(int type)
```

Sets the triggering type to the specified value.

●getPeriod

```
public PSDLTime getPeriod()
```

Returns the period value of this Vertex.

●getFinishWithin

```
public PSDLTime getFinishWithin()
```

Returns the finish within value of this Vertex.

●getMcp

```
public PSDLTime getMcp()
```

Returns the mcp value of this Vertex.

●getMrt

```
public PSDLTime getMrt()
```

Returns the mrt value of this Vertex.

●setPeriod

```
public void setPeriod(PSDLTime p)
```

Sets the period to the specified value.

●setFinishWithin

```
public void setFinishWithin(PSDLTime fw)
```

Sets the finish within to the specified value.

●setMcp

```
public void setMcp(PSDLTime m)
```

Sets the mcp to the specified value.

●setMrt

```
public void setMrt(PSDLTime mr)
```

Sets the mrt to the specified value.

●getImpLanguage

```
public java.lang.String getImpLanguage()
```

Returns the implementation language of this Vertex.

●setImpLanguage

```
public void setImpLanguage(java.lang.String s)
```

Sets the implementation language to the specified value.

●getMetReqmts

```
public java.util.Vector getMetReqmts()
```

Returns the met requirements of this Vertex.

●setMetReqmts

public void **setMetReqmts**(java.util.Vector v)

Sets the met requirements to the specified value.

●getPeriodReqmts

public java.util.Vector **getPeriodReqmts**()

Returns the period requirements of this Vertex.

●setPeriodReqmts

public void **setPeriodReqmts**(java.util.Vector v)

Sets the period reqirements to the specified value.

●getFinishWithinReqmts

public java.util.Vector **getFinishWithinReqmts**()

Returns the finish within requirements of this Vertex.

●setFinishWithinReqmts

public void **setFinishWithinReqmts**(java.util.Vector v)

Sets the finish within requirements to the specified value.

●getMcpReqmts

public java.util.Vector **getMcpReqmts**()

Returns the mcp requirements of this Vertex.

●setMcpReqmts

public void **setMcpReqmts**(java.util.Vector v)

Sets the mcp requirements to the specified value.

●getMrtReqmts

public java.util.Vector **getMrtReqmts**()

Returns the mrt requirements of this Vertex.

●setMrtReqmts

public void **setMrtReqmts**(java.util.Vector v)

Sets the mrt requirements to the specified value.

●getTriggerReqmts

public java.util.Vector **getTriggerReqmts**()

Returns the trigger requirements of this Vertex.

●setTriggerReqmts

public void **setTriggerReqmts**(java.util.Vector v)

Sets the trigger requirements to the specified value.

●getTriggerStreamsList

public java.util.Vector **getTriggerStreamsList**()

Returns the triggering streams of this Vertex.

●setTriggerStreamsList

public void **setTriggerStreamsList**(java.util.Vector v)

Sets the trigger streams list to the specified value.

●getIfCondition

public java.lang.String **getIfCondition**()

Returns the if condition of this Vertex.

●setIfCondition

public void **setIfCondition**(java.lang.String s)

Sets the if condition to the specified value.

●getOutputGuardList

public java.lang.String **getOutputGuardList**()

Returns the output guard list of this Vertex.

●setOutputGuardList

public void setOutputGuardList(java.lang.String s)

Sets the output guard list to the specified value.

●getExceptionGuardList

public java.lang.String getExceptionGuardList()

Returns the exception guard list of this Vertex.

●setExceptionGuardList

public void setExceptionGuardList(java.lang.String s)

Sets the exception guards list to the specified value.

●getExceptionList

public java.lang.String getExceptionList()

Returns the exception list of this Vertex.

●setExceptionList

public void setExceptionList(java.lang.String s)

Sets the exception list to the specified value.

●getTimerOpList

public java.lang.String getTimerOpList()

Returns the timer op list of this Vertex.

●setTimerOpList

public void setTimerOpList(java.lang.String s)

Sets the timer op list to the specified value.

●getInformalDesc

public java.lang.String getInformalDesc()

Returns the informal description of this Vertex.

●setInformalDesc

public void setInformalDesc(java.lang.String s)

Sets the informal description to the specified value.

●getFormalDesc

public java.lang.String getFormalDesc()

Returns the formal description of this Vertex.

●setFormalDesc

public void setFormalDesc(java.lang.String s)

Sets the formal description to the specified value.

●getKeywordList

public java.util.Vector getKeywordList()

Returns the keywords of this Vertex.

●setKeywordList

public void setKeywordList(java.util.Vector v)

Sets the keywords to the specified value.

●getTimerList

public java.util.Vector getTimerList()

Returns the timers of this Vertex.

●setTimerList

public void setTimerList(java.util.Vector v)

Sets the timer list to the specified value.

●getGraphDesc

public java.lang.String getGraphDesc()

Returns the informal graph description of this Vertex.

●setGraphDesc

`public void setGraphDesc(java.lang.String s)`

Sets the graph description to the specified value.

●getGenericList

`public java.lang.String getGenericList()`

Returns the generic list of this Vertex.

●setGenericList

`public void setGenericList(java.lang.String s)`

Sets the generic list to the specified value.

●getSpecReqmts

`public java.util.Vector getSpecReqmts()`

Returns the spec requirements of this Vertex.

●getIntersectionPoint

`public java.awt.Point getIntersectionPoint(java.awt.Point p)`

Returns intersection point of this vertex with the specified point.

●getTerminatorIntersection

`public java.awt.Point getTerminatorIntersection(java.awt.Point p)`

Returns the intersection point of this vertex with the specified point. Called from getIntersectionPoint when this Vertex is a Terminator

●getOperatorIntersection

`public java.awt.Point getOperatorIntersection(java.awt.Point p)`

200

Returns the intersection point of this vertex with the specified point. Called from getIntersectionPoint when this Vertex is an Operator.

●getSpecification

`public java.lang.String getSpecification(boolean hasId)`

Creates the specification construct from its data structures.

Parameters:

hasId - boolean value that specifies if this Vertex has a unique id.

Returns:

returns the string representation of the specification of this Vertex.

●extractString

`public java.lang.String extractString(java.lang.String str, boolean moreSpaces)`

Called from getSpecification. Extracts the string parameter and reformats it to add to the specification.

●extractList

`public java.lang.String extractList(java.util.Vector v)`

Extracts an idList which is represented as a Vector and returns a String representation of the idList so that it will have the form "id1, id2, id3..."

●delete

`public void delete()`

Deletes this Vertex. Deletes all the children of this Vertex and also deletes all the in and out Edges.

APPENDIX D. SOURCE CODE

```
package caps;

import caps.CAPSMain.*;

/**
 * The driver program for CAPS.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class Caps {

/**
 * The constructor for this class.
 *
 * @param args[] The command line parameters.
 * (No command line parameter is necessary for this program.)
 */
public static void main (String args [])
{
    CAPSMainWindow main = new CAPSMainWindow ();
}

} // End of the class Caps
```

```
package caps.CAPSMain;

import javax.swing.JMenuBar;

/**
 * The menubar of the main CAPS window.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class CAPSMainMenuBar extends JMenuBar {

/**
 * The constructor for this class.
 *
 * @param owner The parent class which has declared this menubar.
 */
public CAPSMainMenuBar (CAPSMainWindow owner)
{
    super ();

    // Add the menus
    add (new PrototypeMenu (owner));
    add (new EditMenu (owner));
    add (new DatabasesMenu ());
    add (new ExecSupportMenu ());
    add (new HelpMenu ());
}

} // End of the class CAPSMainMenuBar
```

```java
package caps.CAPSMain;

import java.awt.*;
import javax.swing.*;
import java.io.File;
import caps.Builder.PsdlBuilder;
import caps.Psdl.Vertex;
import caps.Psdl.DataTypes;
import caps.GraphEditor.Editor;
import java.awt.event.*;
import java.util.Vector;
import java.util.Enumeration;

/**
 * The main CAPS window.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class CAPSMainWindow extends JFrame {

    /**
     * The File that contains the PSDL prototype.
     */
    private File prototype;

    /**
     * The Vector that holds references to the open prototypes
     */
    private static Vector openPrototypes;

    /**
     * The constructor for this class.
     */
    public CAPSMainWindow ()
    {
        super ("HSI Designer Mode");        // The title of the frame.

        prototype = null;

        openPrototypes = new Vector (0, 2);

        initialize ();
    }

    /**
     * Initializes the CAPS main window.
     */
    public void initialize ()
    {
        setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
        addWindowListener (new ExitCAPSMain (this));

        /**
         * Places the frame in the upper-right corner of the screen
         */
        Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
        setLocation(screenSize.width - (WIDTH + WIDTH / 2), HEIGHT / 2);

        setResizable (false);

        setJMenuBar (new CAPSMainMenuBar (this));

        JPanel panel = new JPanel ();

        JLabel capsLabel = new JLabel ("Heterogeneous System Integrator");
        capsLabel.setFont (new Font ("Courier", Font.BOLD, 17));
        JLabel imageLabel = new JLabel (new ImageIcon
("caps/Images/caps.gif"));

        panel.add (Box.createHorizontalStrut (5));
        panel.add (imageLabel);
        panel.add (Box.createHorizontalStrut (5));
        panel.add (capsLabel);
        panel.add (Box.createHorizontalStrut (5));

        getContentPane ().add (panel);

        pack ();

        setVisible (true);
    }

    /**
     * Sets the prototype file to the argument.
     *
     * @param f The File that contains the PSDL prototype.
     */
    public void setPrototype (File f)
    {
        prototype = f;
    }

    /**
     * Returns the vector that holds the open prototype files.
     *
     * @return the vector that contains the open prototype files.
     */
    public Vector getOpenPrototypes ()
    {
        return openPrototypes;
    }
```

```java
/**
 * Opens the graphics editor to edit a prototype.
 */
public void editPrototype ()
{
    if (prototype == null) {          // No prototype is selected to
open
        JOptionPane.showMessageDialog (this, "No prototype is selected to
edit.",
                                        "Error Message",
JOptionPane.ERROR_MESSAGE);
    }
    else if (!isPrototypeChanged ()) {        // Attempt to edit the
same prototype.
        JOptionPane.showMessageDialog (this, new String ("Prototype " +
prototype.getName () +
                                        " is already open."),
                                        "Error Message",
JOptionPane.ERROR_MESSAGE);
    }
    else {
        PsdlBuilder.disable_tracing ();       // Disable debug messages
        Vertex root = null;
        root = PsdlBuilder.buildPrototype (prototype);
        if (root == null) {
            root = new Vertex (0, 0, null, false);   // If this is a new
prototype     String name = prototype.getName ();
Prototype name is the same as
            root.setLabel (name.substring (0, name.length () - 5)); // the
file name
        }
        DataTypes types = new DataTypes ();
        types.buildTypes (prototype);
        Editor e = new Editor (prototype, root, types);
        new Thread (e).start ();
        openPrototypes.addElement (e);
    }
}

/**
 * Checks whether or not the current prototype file is already used by
 * a PSDL Editor.
 *
 * @return true if one of the open prototypes is the same as the
current
 * prototype file.
 */
public boolean isPrototypeChanged ()
{
    for (Enumeration enum = openPrototypes.elements ();
enum.hasMoreElements ();) {
        Editor e = (Editor) enum.nextElement ();
        if (prototype.equals (e.getPrototypeFile ()))
            return false;
    }
    return true;
}

/**
 * Removes one element from the openPrototypes vector.
 *
 * @param e the editor that is going to be removed from the vector.
 */
public static void removeEditor (Editor e)
{
    openPrototypes.removeElement (e);
}

/**
 * Checks if the status of any of the open prototypes is
'saveRequired'.
 * Prompts the user to save the prototype.
 *
 * @return true if none of the prototypes need saving.
 */
public boolean isOpenPrototypeSaved ()
{
    boolean flag = true;
    Editor e;
    label :
    for (Enumeration enum = openPrototypes.elements
();enum.hasMoreElements ();) {
        e = (Editor) enum.nextElement ();
        if (e.isSaveRequired ()) {
            int ix = JOptionPane.showConfirmDialog (this, new String
("Save changes to the prototype " +          e.getRoot ().getLabel
() + "?"));
            if (ix == JOptionPane.CANCEL_OPTION) {
                flag = false;
                break label;
            }
            else if (ix == JOptionPane.YES_OPTION)
                e.savePrototype ();
        }
    }
    return flag;
}

} // End of the class CAPSMainWindow
```

```java
package caps.CAPSMain;

import javax.swing.JMenu;
import javax.swing.JMenuItem;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * This class holds the 'Databases' menu items.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class DatabasesMenu extends JMenu implements ActionListener {

    /**
     * Initiates the 'Design Database' event
     */
    private JMenuItem designDBMenuItem = new JMenuItem ("Design Database");

    /**
     * Initiates the 'Software Base' event
     */
    private JMenuItem swBaseMenuItem = new JMenuItem ("Software Base");

    /**
     * Constructor for this class.
     */
    public DatabasesMenu ()
    {
        super ("Databases");

        add (designDBMenuItem);
        add (swBaseMenuItem);

        /*
         * Register the action listeners
         */
        designDBMenuItem.addActionListener (this);
        swBaseMenuItem.addActionListener (this);
    }

    /**
     * Action event handler for the menu events.
     *
     * @param e The action event that is created by selecting a menu item
     from this menu
     */
    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource () == designDBMenuItem) {
            System.out.println ("Design DB has not been implemented yet");
        }
        else if (e.getSource () == swBaseMenuItem) {
            System.out.println ("SW Base has not been implemented yet");
        }
    }

} // End of the class DatabasesMenu
```

```java
package caps.CAPSMain;

import javax.swing.JMenu;
import javax.swing.JMenuItem;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * This class holds the 'Edit' menu items.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class EditMenu extends JMenu implements ActionListener {

    /**
     * Initiates the 'PSDL' event
     */
    private JMenuItem psdlMenuItem = new JMenuItem ("PSDL");

    /**
     * Initiates the 'Ada' event
     */
    private JMenuItem adaMenuItem = new JMenuItem ("Ada");

    /**
     * Initiates the 'Interface' event
     */
    private JMenuItem interfaceMenuItem = new JMenuItem ("Interface");

    /**
     * Initiates the 'Requirements' event
     */
    private JMenuItem requirementsMenuItem = new JMenuItem
("Requirements");

    /**
     * Initiates the 'Change Request' event
     */
    private JMenuItem changeReqMenuItem = new JMenuItem ("Change Request");

    /**
     * Initiates the 'CAPS Defaults' event
     */
    private JMenuItem capsDefaultsMenuItem = new JMenuItem ("HSI Defults");

    /**
     * Initiates the 'Hardware Model' event
     */
    private JMenuItem hwModelMenuItem = new JMenuItem ("Hardware Model");

    /**
     * The main window which owns this menu.
     */
    protected CAPSMainWindow owner;

    /**
     * The constructor for this class.
     *
     * @param f The parent class which has declared this menubar.
     */
    public EditMenu (CAPSMainWindow f)
    {
        super ("Edit");

        owner = f;

        add (psdlMenuItem);
        add (adaMenuItem);
        add (interfaceMenuItem);
        add (requirementsMenuItem);
        add (changeReqMenuItem);
        add (capsDefaultsMenuItem);
        add (hwModelMenuItem);

        /*
         * Register the action listeners
         */
        psdlMenuItem.addActionListener (this);
        adaMenuItem.addActionListener (this);
        interfaceMenuItem.addActionListener (this);
        requirementsMenuItem.addActionListener (this);
        changeReqMenuItem.addActionListener (this);
        capsDefaultsMenuItem.addActionListener (this);
        hwModelMenuItem.addActionListener (this);
    }

    /**
     * Action event handler for the menu events.
     *
     * @param e The action event that is created by selecting a menu item
from this menu
     */
    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource () == psdlMenuItem) {
            owner.editPrototype ();
        }
        else if (e.getSource () == adaMenuItem) {
            System.out.println ("Ada Editor has not been implemented yet");
        }
        else if (e.getSource () == interfaceMenuItem) {
            System.out.println ("Interface Editor has not been implemented
yet");
```

```java
package caps.CAPSMain;

import javax.swing.JMenu;
import javax.swing.JMenuItem;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * This class holds the 'Exec Support' menu items.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class ExecSupportMenu extends JMenu implements ActionListener {

    /**
     * Initiates the 'Translate' event
     */
    private JMenuItem translateMenuItem = new JMenuItem ("Translate");

    /**
     * Initiates the 'Schedule' event
     */
    private JMenuItem scheduleMenuItem = new JMenuItem ("Schedule");

    /**
     * Initiates the 'Compile' event
     */
    private JMenuItem compileMenuItem = new JMenuItem ("Compile");

    /**
     * Initiates the 'Execute' event
     */
    private JMenuItem executeMenuItem = new JMenuItem ("Execute");

    /**
     * Constructor for this class.
     */
    public ExecSupportMenu ()
    {
        super ("Exec Support");

        add (translateMenuItem);
        add (scheduleMenuItem);
        add (compileMenuItem);
        add (executeMenuItem);

        /*
         * Register the action listeners
         */
        translateMenuItem.addActionListener (this);
        scheduleMenuItem.addActionListener (this);
```

```java
        }
        else if (e.getSource () == requirementsMenuItem) {
            System.out.println ("Requirements Editor has not been implemented
yet");
        }
        else if (e.getSource () == changeReqMenuItem) {
            System.out.println ("Change Requirements has not been implemented
yet");
        }
        else if (e.getSource () == capsDefaultsMenuItem) {
            System.out.println ("CAPS Defaults has not been implemented
yet");
        }
        else if (e.getSource () == hwModelMenuItem) {
            System.out.println ("Hardware Model has not been implemented
yet");
        }
    }

} // End of the class EditMenu
```

```java
        compileMenuItem.addActionListener (this);
        executeMenuItem.addActionListener (this);
    }

/**
 * Action event handler for the menu events.
 *
 * @param e The action event that is created by selecting a menu item
from this menu
 */
public void actionPerformed(ActionEvent e)
{
    if (e.getSource () == translateMenuItem) {
        System.out.println ("Translate has not been implemented yet");
    }
    else if (e.getSource () == scheduleMenuItem) {
        System.out.println ("Schedule has not been implemented yet");
    }
    else if (e.getSource () == compileMenuItem) {
        System.out.println ("Compile has not been implemented yet");
    }
    else if (e.getSource () == executeMenuItem) {
        System.out.println ("Execute has not been implemented yet");
    }
}

} // End of the class ExecSupportMenu
```

```java
package caps.CAPSMain;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

/**
 * Closes the caps main window and exits from the program.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
class ExitCAPSMain extends WindowAdapter {

/**
 * The main program that has declared this object
 */
CAPSMainWindow capsMain;

/**
 * The constructor for this class.
 *
 * @param owner The parent class which has declared this menubar.
 */
public ExitCAPSMain (CAPSMainWindow caps)
{
    capsMain = caps;
}

/**
 * Window event handler for the menu events.
 *
 * @param e The window event that is created when the program close
icon is pressed.
 */
public void windowClosing(WindowEvent e)
{
    // Exit the program if the prototypes are saved
    if (capsMain.isOpenPrototypeSaved ())
        System.exit (0);
}

} // End of the class ExitCapsMain
```

```java
package caps.CAPSMain;

import javax.swing.JMenu;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

/**
 * This class implements the 'Help' menu.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class HelpMenu extends JMenu implements ActionListener {

    /**
     * Constructor for this class.
     */
    public HelpMenu ()
    {
        super ("Help");
    }

    /**
     * Action event handler for the menu events.
     *
     * @param e The action event that is created by selecting a menu item
from this menu
     */
    public void actionPerformed(ActionEvent e)
    {
        // Not implemented yet
    }

} // End of the class HelpMenu
```

```java
package caps.CAPSMain;

import javax.swing.*;
import javax.swing.filechooser.FileSystemView;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.util.Vector;

/**
 * This class holds the 'Prototype' menu items.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class PrototypeMenu extends JMenu implements ActionListener {

    /**
     * Initiates the 'New' event
     */
    private JMenuItem newMenuItem = new JMenuItem ("New");

    /**
     * Initiates the 'Open' event
     */
    private JMenuItem openMenuItem = new JMenuItem ("Open");

    /**
     * Initiates the 'Commit Work' event
     */
    private JMenuItem commitWorkMenuItem = new JMenuItem ("Commit Work");

    /**
     * Initiates the 'Retrieve From DDB' event
     */
    private JMenuItem retrieveMenuItem = new JMenuItem ("Retrieve From
DDB");

    /**
     * Initiates the 'Quit' event
     */
    private JMenuItem quitMenuItem = new JMenuItem ("Quit");

    /**
     * The main window which owns this menu.
     */
    protected CAPSMainWindow ownerWindow;

    /**
     * Constructor for this class.
     *
     * @param owner The main window which has created this menu.
```

```java
    */
    public PrototypeMenu (CAPSMainWindow owner)
    {
        super ("Prototype");

        ownerWindow = owner;

        add (newMenuItem);
        add (openMenuItem);
        add (commitWorkMenuItem);
        add (retrieveMenuItem);
        add (quitMenuItem);

        /*
         * Register the action listeners
         */
        newMenuItem.addActionListener (this);
        openMenuItem.addActionListener (this);
        commitWorkMenuItem.addActionListener (this);
        retrieveMenuItem.addActionListener (this);
        quitMenuItem.addActionListener (this);
    }

    /**
     * Action event handler for the menu events.
     *
     * @param e The action event that is created by selecting a menu item
     * from this menu
     */
    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource () == newMenuItem) {
            processNewMenuItem ();
        }
        else if (e.getSource () == openMenuItem) {
            processOpenMenuItem ();
        }
        else if (e.getSource () == commitWorkMenuItem) {
            System.out.println ("Commit Work has not yet been implemented");
        }
        else if (e.getSource () == retrieveMenuItem) {
            System.out.println ("Retrieve has not yet been implemented");
        }
        else if (e.getSource () == quitMenuItem) {
            // Exit the program if all of the prototypes are saved.
            if (ownerWindow.isOpenPrototypeSaved ())
                System.exit (0);
        }
    }

    /**
     * Handles the event which is caused by selecting the 'New' menu item.
     */
    public void processNewMenuItem ()
    {
        // The system property for the home prototype directory.
        String protoHome = System.getProperty ("PROTOTYPEHOME");
        File protoDir;
        if (protoHome == null) {     // If it is not set as a command line
argument
            File homeDir = FileSystemView.getFileSystemView
().getHomeDirectory ();
            protoHome = new String (homeDir + File.separator + ".caps");
            protoDir = new File (protoHome);
            if (!protoDir.exists ())
                protoDir.mkdir ();
        }
        else {
            protoDir = new File (protoHome);
            if (!protoDir.exists ())
                protoDir.mkdir ();
        }

        String proto = JOptionPane.showInputDialog (ownerWindow,
                                "Create new prototype : ",  "New",
JOptionPane.PLAIN_MESSAGE);
        if (proto == null)
            return;
        else if (proto.indexOf (File.separator) == -1)
            JOptionPane.showMessageDialog (ownerWindow, "Please enter the
version number with the prototype name",          "Error Message",
JOptionPane.ERROR_MESSAGE);
        else {
            String name = proto.substring (0, proto.indexOf
(File.separator));
            String version = proto.substring (proto.indexOf (File.separator),
proto.length ());
            File file = new File (protoHome + File.separator + proto +
File.separator +                    name + ".psdl");

            if (file.exists ()) {
                int selected = JOptionPane.showConfirmDialog (ownerWindow,
"Selected prototype file already exists.\n" +
                                                 "Do you want to
overwrite it ?");
                if (selected == JOptionPane.YES_OPTION) {
                    try {
                        file.delete ();
                        file.createNewFile ();
                    } catch (java.io.IOException ex) {
                        System.out.println (ex);
                    }
                }
                ownerWindow.setPrototype (file);
```

```java
            }
        }
        else {
            try {
                File dir = file.getParentFile ().getParentFile ();
                dir.mkdir ();
                File vers = file.getParentFile ();
                vers.mkdir ();
                file.createNewFile ();
            } catch (java.io.IOException ex) {
                System.out.println (ex);
            }
        }
        ownerWindow.setPrototype (file);
    }
}

/**
 * Handles the event which is caused by selecting the 'Open' menu
item.
 */
public void processOpenMenuItem ()
{
    String protoHome = System.getProperty ("PROTOTYPEHOME");
    File protoDir;
    if (protoHome == null) {   // If it is not set as a command line
argument
        File homeDir = FileSystemView.getFileSystemView
().getHomeDirectory ();
        protoHome = new String (homeDir + File.separator + ".caps");
        protoDir = new File (protoHome);
        if (!protoDir.exists ())
            protoDir.mkdir ();
    }
    else {
        protoDir = new File (protoHome);
        if (!protoDir.exists ())
            protoDir.mkdir ();
    }

    Vector prototypeNames = new Vector (0, 2);
    File [] dirs = protoDir.listFiles ();
    String protoName = "";

    if (dirs.length == 0) {
        JOptionPane.showMessageDialog (ownerWindow, "No prototype is is
found to open",
                                       "Error Message",
JOptionPane.ERROR_MESSAGE);
    }
    else {
        for (int ix = 0; ix < dirs.length; ix++) {
            protoName = dirs [ix].getName ();
            File subDirs [] = dirs [ix].listFiles ();
            for (int jx = 0; jx < subDirs.length; jx++) {
                prototypeNames.addElement (protoName.concat (File.separator
+ subDirs [jx].getName ()));
            }
        }

        Object [] protos = prototypeNames.toArray ();
        String selected = (String) JOptionPane.showInputDialog
(ownerWindow, "Select a prototype : ",
                            "Open", JOptionPane.INFORMATION_MESSAGE,
null, protos, protos [0]);

        if (selected != null) {
            File selectedDir = new File (protoHome + File.separator +
selected);
            File file = new File (selectedDir.getAbsolutePath () +
File.separator +
                                selectedDir.getParentFile ().getName ()
+ ".psdl");
            if (!file.exists ())
                JOptionPane.showMessageDialog (ownerWindow, "The selected
prototype file cannot be opened",
                                    "Error Message",
JOptionPane.ERROR_MESSAGE);
        }
    }
}

} // End of the class PrototypeMenu
```

211

```java
package caps.Display;

import java.awt.*;
import java.awt.font.*;
import java.awt.geom.*;
import javax.swing.JLabel;
import caps.Psdl.DataFlowComponent;
import java.util.Vector;

/**
 *
 * This is an abstract super class of EdgePath and DisplayVertex.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public abstract class DisplayComponent {

    /**
     * The size of the Handles.
     */
    public static final int HANDLESIZE = 6;

    /**
     * The DataFlowComponent that this object associates with.
     */
    protected DataFlowComponent dfc;

    /**
     * The shape of the label of the component.
     */
    TextLayout labelShape;

    /**
     * The shape of the met of the component .
     */
    TextLayout metShape;

    /**
     * The constructor is protected so it cannot be instantiated directly.
     *
     * @param d the DataFlowComponent that is associated with this object.
     */
    protected DisplayComponent (DataFlowComponent d)
    {
        dfc = d;
        labelShape = null;
        metShape = null;
    }

    /**
     * This abstract method is implemented in subclasses.
     */
    public abstract Shape getShape ();

    /**
     * This abstract method is implemented in subclasses.
     */
    public abstract boolean containsClickedPoint (int xLoc, int yLoc);

    /**
     * This abstract method is implemented in subclasses.
     */
    public abstract Vector getHandles ();

    /**
     * This abstract method is implemented in subclasses.
     */
    public abstract void update ();

    /**
     * This abstract method is implemented in subclasses.
     */
    public abstract void delete ();

    /**
     * Gets the label from the DataFlowComponent and creates a TextLayout
     * shape for the label.
     *
     * @param g2D the graphics context of the DrawPanel
     */
    public void setLabelShape (Graphics2D g2D)
    {
        labelShape = new TextLayout (dfc.getLabel (), dfc.getlFont (),
g2D.getFontRenderContext ());
    }

    /**
     * Returns the bounding rectangle of the label shape.
     *
     * @return the bounding rectangle of the label shape.
     */
    public Rectangle2D getLabelShapeBounds ()
    {
        Rectangle2D r2D = labelShape.getBounds ();
        int x = dfc.getX () + dfc.getLabelXOffset () - (int)
labelShape.getBounds ().getWidth () / 2;
        int y = dfc.getY () + dfc.getLabelYOffset () - (int)
labelShape.getBounds ().getHeight () / 2;
        r2D.setRect (x, y, r2D.getWidth (), r2D.getHeight ());
        return r2D;
    }

    /**
     * This abstract method is implemented in subclasses.
     */
```

```
 * Gets the location of the label shape and draws it into the
DrawPanel.
 *
 * @param g2D the graphics context of the DrawPanel.
 */
public void drawLabelShape (Graphics2D g2D)
{
    int x = dfc.getX () + dfc.getLabelXOffset () - (int)
labelShape.getBounds ().getWidth () / 2;
    int y = dfc.getY () + dfc.getLabelYOffset () + (int)
labelShape.getBounds ().getHeight () / 2;
    labelShape.draw (g2D, x, y);
}

/**
 * Creates a vector that holds the handles of a string (met or label).
 *
 * @param r2D the bounding rectangle of the string.
 * @return returns the Vector that holds the handles.
 */
public Vector getStringHandles (Rectangle2D r2D)
{
    Vector v = new Vector ();
    int i = HANDLESIZE / 2;
    v.add (new Rectangle2D.Double (r2D.getMinX () - i, r2D.getMinY () -
i, HANDLESIZE, HANDLESIZE));
    v.add (new Rectangle2D.Double (r2D.getMaxX () - i, r2D.getMinY () -
i, HANDLESIZE, HANDLESIZE));
    v.add (new Rectangle2D.Double (r2D.getMinX () - i, r2D.getMaxY () -
i, HANDLESIZE, HANDLESIZE));
    v.add (new Rectangle2D.Double (r2D.getMaxX () - i, r2D.getMaxY () -
i, HANDLESIZE, HANDLESIZE));
    return v;
}

/**
 * Gets the met (or latency) from the DataFlowComponent and creates a
TextLayout shape for the met.
 *
 * @param g2D the graphics context of the DrawPanel
 */
public void setMetShape (Graphics2D g2D)
{
    if (dfc.getMet () != null)     // It may not have an met
        metShape = new TextLayout (dfc.getMet ().toString (),
dfc.getMet1Font (), g2D.getFontRenderContext ());
    else
        metShape = new TextLayout (" ", dfc.getMet1Font (),
g2D.getFontRenderContext ());
}

/**

 * Returns the bounding rectangle of the met (or latency) shape.
 *
 * @return the bounding rectangle of the met (or latency) shape.
 */
public Rectangle2D getMetShapeBounds ()
{
    Rectangle2D r2D = metShape.getBounds ();
    int x = dfc.getX () + dfc.getMetXOffset () - (int)
metShape.getBounds ().getWidth () / 2;
    int y = dfc.getY () + dfc.getMetYOffset () - (int)
metShape.getBounds ().getHeight () / 2;
    r2D.setRect (x, y, r2D.getWidth (), r2D.getHeight ());
    return r2D;
}

/**
 * Gets the location of the met (or latency) shape and draws it into
the DrawPanel.
 *
 * @param g2D the graphics context of the DrawPanel.
 */
public void drawMetShape (Graphics2D g2D)
{
    if (dfc.getMet () != null) {
        int x = dfc.getX () + dfc.getMetXOffset () - (int)
metShape.getBounds ().getWidth () / 2;
        int y = dfc.getY () + dfc.getMetYOffset () + (int)
metShape.getBounds ().getHeight () / 2;
        metShape.draw (g2D, x, y);
    }
}

/**
 * Returns the DataFlowComponent that is associated with this object.
 *
 * @return the DataFlowComponent that is associated with this object.
 */
public DataFlowComponent getDataFlowComponent ()
{
    return dfc;
}

} // End of the class DisplayComponent
```

```java
package caps.Display;

import java.awt.geom.*;
import java.awt.*;
import java.awt.font.*;
import caps.Psdl.*;
import java.util.Vector;

/**
 * An instance of this class is created when external streams are
created.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class DisplayExternal extends DisplayComponent {

    /**
     * The External object that is associated with this object.
     */
    protected External external;

    /**
     * The shape of the External.
     */
    protected Rectangle2D.Double shape;

    /**
     * The constructor for this class.
     *
     * @param e the External that is associated with this object.
     */
    public DisplayExternal (External e)
    {
        super (e);
        external = e;
        shape = new Rectangle2D.Double (e.getX (), e.getY (), 0.5, 0.5);
    }

    /**
     * Sets the location of this shape on the DrawPanel
     */
    public void setLocation ()
    {
        double x = external.getX ();
        double y = external.getY ();
        shape.setFrame (x, y, shape.getWidth (), shape.getHeight ());
    }

    /**
     * Updates the location and the width of this shape.
     */
    public void update ()
    {
        setLocation ();
    }

    /**
     * Always returns false since the shape is not displayed in the
DrawPanel.
     *
     * @param xLoc the x location of the clicked point.
     * @param yLoc the y location of the clicked point.
     * @return false.
     */
    public boolean containsClickedPoint (int xLoc, int yLoc)
    {
        return false;
    }

    /**
     * Returns the vector that contains the handles of the shape.
     *
     * @return an empty Vector.
     */
    public Vector getHandles ()
    {
        Vector v = new Vector ();
        return v;
    }

    /**
     * Returns the shape that represents the External.
     *
     * @return the shape that represents the External.
     */
    public Shape getShape ()
    {
        return (Shape) shape;
    }

    /**
     * Deletes the external that is associated with this object.
     */
    public void delete ()
    {
        external.delete ();
        external = null;
        shape = null;
    }

} // End of the class DisplayExternal
```

```java
package caps.Display;

import java.awt.geom.*;
import java.awt.*;
import java.awt.font.*;
import caps.Psdl.*;
import java.util.Vector;

/**
 * This class holds a shape for its associated Vertex.
 * It can either be a rectangle for terminators or it can be a circle
 * for the operators.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class DisplayVertex extends DisplayComponent {

    /**
     * The Vertex that is associated with this object.
     */
    protected Vertex vertex;

    /**
     * The shape of the Vertex.
     */
    protected RectangularShape shape;

    /**
     * The constructor for this class.
     *
     * param v the Vertex that is associated with this object.
     */
    public DisplayVertex (Vertex v)
    {
        super (v);
        vertex = v;
        setShape ();
        setWidth ();
        setLocation ();
    }

    /**
     * Sets the location of this shape on the DrawPanel
     */
    public void setLocation ()
    {
        // x and y represent the upper left corner of the shape
        double x = vertex.getX () - shape.getWidth () / 2;
        double y = vertex.getY () - shape.getHeight () / 2;
        shape.setFrame (x, y, shape.getWidth (), shape.getHeight ());
    }

    /**
     * Sets the width of this shape.
     */
    public void setWidth ()
    {
        double width = vertex.getWidth ();
        double height = vertex.getHeight ();
        shape.setFrame (vertex.getX () - shape.getWidth () / 2, vertex.getY
() - shape.getHeight () / 2,
                width, height);
    }

    /**
     * Updates the location and the width of this shape.
     */
    public void update ()
    {
        setLocation ();
        setWidth ();
    }

    /**
     * Checks whether the bounding box of the shape contains the the
location where the mouse is clicked.
     *
     * @param xLoc the x location of the clicked point.
     * @param yLoc the y location of the clicked point.
     * @return true if the bounding box contains the clicked point.
     */
    public boolean containsClickedPoint (int xLoc, int yLoc)
    {
        return getShape ().contains (new Point (xLoc, yLoc));
    }

    /**
     * Returns the vector that contains the handles of the shape.
     *
     * @return the vector that contains the handles of the shape
     */
    public Vector getHandles ()
    {
        Vector v = new Vector ();
        RectangularShape s = (RectangularShape) getShape ();
        int i = HANDLESIZE / 2;
        v.add (new Rectangle2D.Double (s.getMinX () - i, s.getMinY () - i,
HANDLESIZE, HANDLESIZE));
        v.add (new Rectangle2D.Double (s.getMaxX () - i, s.getMinY () - i,
HANDLESIZE, HANDLESIZE));
        v.add (new Rectangle2D.Double (s.getMinX () - i, s.getMaxY () - i,
HANDLESIZE, HANDLESIZE));
```

215

```java
        v.add (new Rectangle2D.Double (s.getMaxX () - i, s.getMaxY () - i,
HANDLESIZE, HANDLESIZE));
        return v;
    }

    /**
     * Sets the shape of this object to a circle if the associated Vertex
is an operator
     * or sets it to a rectangle if the Vertex is a Terminator
     */
    public void setShape ()
    {
        if (vertex.isTerminator ())
            shape = new Rectangle2D.Double ();
        else
            shape = new Ellipse2D.Double ();
    }

    /**
     * Returns the shape that represents the Vertex.
     *
     * @return the shape that represents the Vertex.
     */
    public Shape getShape ()
    {
        return (Shape) shape;
    }

    /**
     * This method is called if the Vertex is composite. It calculates and
returns
     * a smaller inner shape.
     *
     * @return the inner shape for the composite Vertex.
     */
    public Shape getInnerShape ()
    {
        if (vertex.isTerminator ())
            return (Shape) new Rectangle2D.Double (shape.getX () + 4,
shape.getY () + 4,
                                   shape.getWidth () - 8,
shape.getHeight () - 8);
        else
            return (Shape) new Ellipse2D.Double (shape.getX () + 4,
shape.getY () + 4,
                                   shape.getWidth () - 8,
shape.getHeight () - 8);
    }

    /**
     * Returns a shape that is slightly smaller than the shape of this
object.
     * The shape that is returned will be painted with the color of the
Vertex.
     *
     * @return a shape that is slightly smaller than the shape of th
object.
     */
    public Shape getPaintedShape ()
    {
        if (vertex.isTerminator ()) {
            if (vertex.isLeaf ())
                return (Shape) new Rectangle2D.Double (shape.getX () + 1,
shape.getY () + 1,                 shape.getWidth () - 1.0f,
shape.getHeight () - 1.0f);
            else
                return (Shape) new Rectangle2D.Double (shape.getX () + 5,
shape.getY () + 5,                 shape.getWidth () - 10,
shape.getHeight () - 10);
        }
        else {
            if (vertex.isLeaf ())
                return (Shape) new Ellipse2D.Double (shape.getX () + 1f,
shape.getY () + 1f,                shape.getWidth () - 2f,
shape.getHeight () - 2f);
            else
                return (Shape) new Ellipse2D.Double (shape.getX () + 5,
shape.getY () + 5,                 shape.getWidth () - 10,
shape.getHeight () - 10);
        }
    }

    /**
     * Deletes the vertex that is associated with this object.
     */
    public void delete ()
    {
        vertex.delete ();
        vertex = null;
        shape = null;
    }

} // End of the class DisplayVertex
```

```java
package caps.Display;

import caps.Psdl.Edge;
import java.util.*;
import java.awt.*;
import java.awt.geom.*;

/**
 * This class represents an Edge on the DrawPanel.
 * It contains a GeneralPath shape to represent the Edge.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class EdgePath extends DisplayComponent {

    /**
     * The Edge that is associated with this object.
     */
    protected Edge edge;

    /**
     * The shape of the Edge.
     */
    protected GeneralPath shape;

    /**
     * The constructor for this class.
     *
     * @param e the Edge that is associated with this object.
     */
    public EdgePath (Edge e)
    {
        super (e);
        edge = e;
        shape = new GeneralPath ();
    }

    /**
     * Returns the shape that represents the Edge.
     *
     * @return the shape that represents the Edge.
     */
    public Shape getShape ()
    {
        return (Shape) shape;
    }

    /**
     * Checks whether the bounding box of the shape contains the the
     * location where the mouse is clicked.
     *
     * @param xLoc the x location of the clicked point.
     * @param yLoc the y location of the clicked point.
     * @return true if the bounding box contains the clicked point.
     */
    public boolean containsClickedPoint (int xLoc, int yLoc)
    {
        int HITDISTANCE = 10;
        Vector points = edge.getPoints ();
        for (Enumeration enum = points.elements (); enum.hasMoreElements
();) {
            Point p = (Point) enum.nextElement ();
            if (p != points.firstElement () && p != points.lastElement ())
                enum.nextElement ();             // Waste the other point
            if ((Math.abs (p.x - xLoc) <= HITDISTANCE) && (Math.abs (p.y -
yLoc) <= HITDISTANCE)) {
                edge.setSelectedHandleIndex (points.indexOf (p));
                return true;
            }
        }
        return false;
    }

    /**
     * Updates the shape by polling values from the associated Edge
object.
     */
    public void update ()
    {
        edge.correctEndingPoints ();
        Vector points = edge.getPoints ();
        Point p1;
        Point p2;
        shape.reset ();
        for (Enumeration enum = points.elements ();enum.hasMoreElements ();) {
            p1 = (Point) enum.nextElement ();
            if (p1.equals (points.firstElement ())) {
                shape.moveTo (p1.x, p1.y);
                p2 = (Point) enum.nextElement ();
            }
            else if (p1.equals (points.lastElement ())) {
                p2 = p1;
            }
            else {
                p2 = (Point) enum.nextElement ();
            }
            shape.quadTo (p1.x, p1.y, p2.x, p2.y);
        }
        p2 = (Point) points.lastElement ();
        p1 = (Point) points.elementAt (points.size () - 2);
        buildArrowHead (p2, p1);
    }
```

217

```java
/**
 * Creates an arrow head for the stream.
 *
 * @param last the point before the ending point of the stream.
 * @param end the last point of the stream.
 */
public void buildArrowHead (Point last, Point end)
{
    double ARROWANGLE = 25.0;
    double ARROWSIDELENGTH = 15.0;
    double angle, tempAngle;
    double halfArrowAngle = ARROWANGLE / 2.0 * Math.PI / 180.0;

    if (last.x == end.x) {
        if (last.y > end.y)
            angle = Math.PI / 2.0;
        else
            angle = 3.0 * Math.PI / 2;
    }
    else {
        angle = Math.atan ((double) (last.y - end.y) / (double) (last.x -
end.x));

        if (last.x < end.x)
            angle = Math.PI + angle;
    }

    tempAngle = angle - halfArrowAngle;
    shape.lineTo (last.x - (int) (Math.cos (tempAngle) *
ARROWSIDELENGTH),    last.y - (int) (Math.sin(tempAngle) *
ARROWSIDELENGTH));
    tempAngle = angle + halfArrowAngle;
    shape.lineTo (last.x - (int) (Math.cos (tempAngle) *
ARROWSIDELENGTH),    last.y - (int) (Math.sin(tempAngle) *
ARROWSIDELENGTH));
    shape.lineTo (last.x, last.y);
}

/**
 * Returns the vector that contains the handles of the shape.
 *
 * @return the vector that contains the handles of the shape
 */
public Vector getHandles ()
{
    Vector v = new Vector ();
    Vector points = edge.getPoints ();
    Point p;
    int i = HANDLESIZE / 2;
    for (Enumeration enum = points.elements (); enum.hasMoreElements
();) {
        p = (Point) enum.nextElement ();
        if (p != points.firstElement () && p != points.lastElement ())
            enum.nextElement ();    // Waste the other point
            v.add (new Rectangle2D.Double (p.x - i, p.y - i, HANDLESIZE,
HANDLESIZE));
    }

    return v;
}

/**
 * Deletes the Edge that is associated with this object.
 */
public void delete ()
{
    edge.delete ();
    edge = null;
    shape = null;
}

} // End of the class EdgePath
```

218

```java
package caps.GraphEditor;

import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;
import java.awt.event.*;
import java.awt.print.*;
import java.util.*;
import caps.Psdl.*;
import caps.Display.*;

/**
 * The drawpanel is the place where the prototypes are
 * drawn on the screen.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class DrawPanel extends JPanel implements MouseListener,
MouseMotionListener, ActionListener {

    /**
     * The constant width of the DrawPanel
     */
    public static final int WIDTH = 1024;

    /**
     * The constant height of the DrawPanel
     */
    public static final int HEIGHT = 768;

    private final Cursor DEFAULT_CURSOR = new Cursor
(Cursor.DEFAULT_CURSOR);

    private final Cursor HAND_CURSOR = new Cursor (Cursor.HAND_CURSOR);

    private final Cursor MOVE_CURSOR = new Cursor (Cursor.MOVE_CURSOR);

    /**
     * The constant which specifies an operator
     */
    public final static int OPERATOR = 1;

    /**
     * The constant which specifies a terminator
     */
    public final static int TERMINATOR = 2;

    /**
     * The constant which specifies a stream
     */
    public final static int STREAM = 3;
```

```java
package caps.GraphEditor;

public class ColorConstants {

    public static String COLOR_NAMES [] = { "Aqua marine", "Black", "Blue",
"Blue violet", "Brown",
    "Cadet blue", "Coral", "Cornflower blue", "Cyan", "Dark green",
"Dark olive green",
    "Dark orchid", "Dark slate blue", "Dark slate gray", "Dark
turquoise", "Dim gray",
    "Fire brick", "Forest green", "Gold", "Golden rod", "Grey",
"Green", "Green Yellow",
    "Indian red", "Khaki", "Light blue", "Light grey", "Light steel
blue", "Lime green",
    "Magenta", "Maroon", "Medium aqua marine", "Medium blue", "Medium
orchid", "Medium sea green",
    "Medium slate blue", "Medium spring green", "Medium turquoise",
"Medium violet red",
    "Midnight blue", "Navy blue", "Orange", "Orange red", "Orchid",
"Pale green", "Pink", "Plum",
    "Red", "Salmon", "Sea green", "Sienna", "Sky blue", "Slate blue",
"Spring green", "Steel blue",
    "Tan", "Thistle", "Turquoise", "Violet", "Violet red", "Wheat",
"White", "Yellow", "Yellow green" };

    public static int RGB_VALUES [] = {7396243, 0, 255, 10444703, 10889770,
6266783, 16744192,
    4342383, 65535, 3100463, 5197615, 10040013, 7021454, 3100495,
7377883, 5526612, 9315107,
    2330147, 13467442, 14408560, 12632256, 65280, 9689968, 5123887,
10461023, 12638681,
    11053224, 9408445, 3329330, 16711935, 9315179, 3329433, 3289805,
9662683, 4353858,
    8323327, 8388352, 7396315, 14381203, 3092303, 2302862, 16744192,
16720896, 14381275,
    9419919, 12357519, 15379946, 16711680, 7291458, 2330216, 9333539,
3316172, 65407,
    2321294, 14390128, 14204888, 11397866, 5189455, 13382297,
1421126263, 16777215, 16776960,
    10079282 };

} // End of the class ColorConstants
```

```java
/**
 * The value of this variable is true if the toolbar is in the select
mode
 */
protected boolean selectMode;

/**
 * The frame which has created this DrawPanel object
 */
protected Editor parentFrame;

/**
 * This vector holds the shapes that are drawn in the DrawPanel.
 * Each shape is redrawn in the paint method by polling them from this
Vector.
 */
protected Vector displayComponentVector;

protected Vector handlesVector;

protected DisplayComponent selectedComponent;

protected boolean MOVING_COMPONENT = false;

protected boolean MOVING_LABEL = false;

protected boolean MOVING_MET = false;

protected boolean RESIZING = false;

protected boolean IS_COLLECTING_POINTS = false;

protected boolean MOVING_ALL = false;

protected Point2D diagonalPoint;

protected VertexProperties vPropertyPanel;

protected EdgeProperties ePropertyPanel;

protected Vertex parentVertex;            // The parent of the current
Level

protected EdgePath currentEdge;

protected boolean selectionDefault;

/**
 * Current component is either an OPERATOR, or a TERMINATOR or a
STREAM
 * according to the selection from the toolbar.
 */
protected int currentComponent;

protected Popup popupMenu;

protected boolean selectAllMode;

protected Point prevPoint;

protected Rectangle bounds;

protected int currentColor;

protected int currentFont;

/**
 * Constructs a new ToolBar object
 *
 * @param frame The parent frame of this DrawPanel object.
 */
public DrawPanel (Editor frame, Vertex root)
{
    super ();

    popupMenu = new Popup (this);

    parentFrame = frame;

    setAlignmentX (LEFT_ALIGNMENT);                      // Panel does not
accept these
    setAlignmentY (TOP_ALIGNMENT);
    setBorder (BorderFactory.createEtchedBorder ());

    selectMode = true;                          // Initially in the
selectMode.
    selectedComponent = null;
    currentEdge = null;

    vPropertyPanel = new VertexProperties (frame);
    ePropertyPanel = new EdgeProperties (frame);
    vPropertyPanel.setVisible (false);
    ePropertyPanel.setVisible (false);

    TextEditor editor = new TextEditor (frame);
    IdListEditor idEditor = new IdListEditor (frame);

    displayComponentVector = new Vector ();
    handlesVector = new Vector ();

    parentVertex = root;        // This is the root

    diagonalPoint = null;
```

```java
        setCursor (DEFAULT_CURSOR);

        selectAllMode = false;
        selectionDefault = false;

        prevPoint = new Point ();

        bounds = null;

        currentColor = 61;    // White; 61 because it is index
        currentFont = 4;      // Courier Plain 12    (it is 5 - 1 )

        addMouseListener (this);              // Register mouse events
        addMouseMotionListener (this);
    }

    // Pending should do more things (erase handles, etc)
    /**
     * Sets the select mode to true or false. The panel is generally in
     * the select mode unless another button is pressed in the toolbar.
     *
     * @param mode true if the panel is going to be in the select mode.
     */
    public void setSelectMode (boolean mode)
    {
        selectMode = mode;
        if (selectMode == false && selectedComponent != null) {
            selectedComponent = null;
            eraseHandles ();
        }
    }

    public void gotoRoot ()
    {
        if (parentVertex.isRoot () == false)
            setParentVertex ((Vertex) parentVertex.getRoot (), null);
        parentFrame.getToolBar ().setOperatorButton (true);
    }

    public void gotoParent ()
    {
        if (parentVertex.isRoot () == false)      // If this is
not the root of this tree
            setParentVertex ((Vertex) parentVertex.getParent (), null);
        parentFrame.getToolBar ().setOperatorButton (true);
    }

    public void decompose ()
    {
        if (selectedComponent == null)
            return;

        DataFlowComponent dfc = selectedComponent.getDataFlowComponent ();
        if (dfc instanceof Vertex && !((Vertex) dfc).isTerminator ())
            parentFrame.getToolBar ().setOperatorButton (false);
        if (selectedComponent instanceof DisplayVertex && selectedComponent
!= null) {
            //selectedComponent.getDataFlowComponent ().setAllowsChildren
(true);
            setParentVertex ((Vertex) selectedComponent.getDataFlowComponent
(), null);
        }
    }

    // Invoked from the treepanel
    public void changeLevel (Vertex parent)
    {
        setParentVertex (parent, null);
        if (parent.isTerminator ())
            parentFrame.getToolBar ().setOperatorButton (false);
        else
            parentFrame.getToolBar ().setOperatorButton (true);
    }

    public void setParentVertex (Vertex v, Graphics2D g2D)
    {
        setSelectMode (false);
        parentVertex = v;
        if (g2D == null)
            g2D = (Graphics2D) getGraphics ();
        displayComponentVector.removeAllElements ();
        for (Enumeration enum = parentVertex.children ();
enum.hasMoreElements ();) {
            DataFlowComponent dfc = (DataFlowComponent) enum.nextElement ();
            if (dfc instanceof Vertex && !((Vertex) dfc).isTerminator ()) &&
                !(dfc instanceof External)) {
                DisplayVertex op = new DisplayVertex ((Vertex) dfc);
                op.setLabelShape (g2D);
                op.setMetShape (g2D);
                displayComponentVector.addElement (op);
            }
            else if (dfc instanceof Vertex && (((Vertex) dfc).isTerminator
()) &&
                !(dfc instanceof External)) {
                DisplayVertex tr = new DisplayVertex ((Vertex) dfc);
                tr.setLabelShape (g2D);
                tr.setMetShape (g2D);
                displayComponentVector.addElement (tr);
            }
            else {
                EdgePath ep = new EdgePath ((Edge) dfc);
                ep.setLabelShape (g2D);
                ep.setMetShape (g2D);
                ((Edge) ep.getDataFlowComponent ()).correctLabelOffset ();
```

221

```java
        ep.update ();
        displayComponentVector.addElement (ep);
        if (((Edge) dfc).getSource () instanceof External) {
            DisplayExternal extern = new DisplayExternal
((External)((Edge) dfc).getSource ());
            extern.setLabelShape ((Graphics2D) getGraphics ());
            extern.setMetShape ((Graphics2D) getGraphics ());
            // *** Maybe we don't need this *******
            displayComponentVector.addElement (extern);
        }
        else if (((Edge) dfc).getDestination () instanceof External) {
            DisplayExternal extern = new DisplayExternal
((External)((Edge) dfc).getDestination ());
            extern.setLabelShape ((Graphics2D) getGraphics ());
            extern.setMetShape ((Graphics2D) getGraphics ());
            // *** Maybe we don't need this *******
            displayComponentVector.addElement (extern);
        }
    }
    clearAllComponentsFromScreen (g2D);
    paint (g2D);
    setSelectMode (true);
}

public void eraseHandles ()
{
    System.out.println ("Inside eraseHandles");
    //Graphics2D g2D = (Graphics2D) getGraphics ();
    //for (Enumeration e = handlesVector.elements (); e.hasMoreElements
();) {
    //    g2D.setColor (Color.white);
    //    g2D.fill ((Shape) e.nextElement ());
    //}
    handlesVector.removeAllElements ();
    clearAllComponentsFromScreen (null);
    paint (getGraphics ());
}

public void clearAllComponentsFromScreen (Graphics2D g2D)
{
    if (g2D == null)
        g2D = (Graphics2D) getGraphics ();
    g2D.setColor (Color.white);
    g2D.fillRect (0, 0, WIDTH, HEIGHT);
}

/**
* Sets the currentComponent variable to the specified argument.
*
* @param component OPERATOR, TERMINATOR or STREAM
*/
public void setCurrentComponent (int component)
{
    currentComponent = component;
}

public void setSelectedDFC (DataFlowComponent dfc)
{
    DisplayComponent dc;
    for (Enumeration enum = displayComponentVector.elements ();
enum.hasMoreElements ();) {
        dc = (DisplayComponent) enum.nextElement ();
        if (dc.getDataFlowComponent ().equals (dfc)) {
            selectedComponent = dc;
            handlesVector = dc.getHandles ();
            System.out.println (selectedComponent);
            paint (getGraphics ());
        }
    }
}

// ********** Pending drawlist ?? ***********
/**
* Creates a new Operator and a new OperatorCircle object.
* Calls the paintComponent () method to draw the component to this
panel.
*
* @param xLoc The x location of the component.
* @param yLoc The y location of the component.
*/
public void processOperator (int xLoc, int yLoc)
{
    Graphics2D g2D = (Graphics2D) getGraphics ();
    if (selectionDefault)
        setSelectMode (true);    // It will allow to place only one
component at a time
        //System.out.println (selectMode);
        Vertex op = new Vertex (xLoc, yLoc, parentVertex, false);
        op.setColor (currentColor + 1);
        op.setLabelFontIndex (currentFont + 1);
        op.setMetFontIndex (currentFont + 1);
        parentFrame.getTreePanel ().addNewDFC (op, parentVertex);
        DisplayVertex opCircle = new DisplayVertex (op);
        opCircle.setLabelShape (g2D);
        opCircle.setMetShape (g2D);    // *** Maybe we don't need this
*******
        paintComponent (opCircle);
        displayComponentVector.addElement (opCircle);
}

// ********** Pending drawlist ?? ***********
/**
* Creates a new Terminator and a new TerminatorRectangle object.
```

222

```
 * Calls the paintComponent () method to draw the component to this
panel.
 *
 * @param xLoc The x location of the component.
 * @param yLoc The y location of the component.
 */
public void processTerminator (int xLoc, int yLoc)
{
    Graphics2D g2D = (Graphics2D) getGraphics ();
    if (selectionDefault)
        setSelectMode (true);        // It will allow to place only one
component at a time
    Vertex term = new Vertex (xLoc, yLoc, parentVertex, true);
    term.setColor (currentColor + 1);
    term.setLabelFontIndex (currentFont + 1);
    term.setMetFontIndex (currentFont + 1);
    parentFrame.getTreePanel ().addNewDFC (term, parentVertex);
    DisplayVertex tRectangle = new DisplayVertex (term);
    tRectangle.setLabelShape (g2D);
    tRectangle.setMetShape (g2D);
    paintComponent (tRectangle);
    displayComponentVector.addElement (tRectangle);
}

public void processStream (int x, int y, int clicks)
{
    DisplayComponent dc;
    Vertex v = null;
    for (Enumeration enum = displayComponentVector.elements ();
enum.hasMoreElements ();) {
        dc = (DisplayComponent) enum.nextElement ();
        //if (dc instanceof DisplayVertex) {      // *** Pending needs to
check if it is a stream ***
        if (dc instanceof DisplayVertex && dc.getShape ().contains (x,
y))
        {
            v = (Vertex) dc.getDataFlowComponent ();
        //}
    }
    //System.out.println (v.getLabel ());     // **** Returns OKAY
    if (IS_COLLECTING_POINTS) {        // Second or more click
        System.out.println ("Collecting points");
        if (v != null) {      // Found the destination
            System.out.println ("The destination is" + v.getLabel ());
            v.addInEdge ((Edge) currentEdge.getDataFlowComponent ());
            ((Edge) currentEdge.getDataFlowComponent ()).setDestination
(v);
            ((Edge) currentEdge.getDataFlowComponent ()).addPoint (x, y);
            parentFrame.getTreePanel ().addNewDFC ((Edge)
currentEdge.getDataFlowComponent (), parentVertex);
            currentEdge.setLabelShape ((Graphics2D) getGraphics ());
            currentEdge.setMetShape ((Graphics2D) getGraphics ());

            ((Edge) currentEdge.getDataFlowComponent
()).correctLabelOffset ();
            currentEdge.update ();
            IS_COLLECTING_POINTS = false;
            displayComponentVector.addElement (currentEdge);
            if (selectionDefault) {
                parentFrame.getToolBar ().enableSelectButton ();
                setSelectMode (true);
            }
            paintComponent (currentEdge);
        }
        else {      // collect the next point
            ((Edge) currentEdge.getDataFlowComponent ()).addPoint (x, y);
        }
    }
    else if (v == null) {      // This is an external -> vertex stream
        IS_COLLECTING_POINTS = true;
        External ex = new External (x, y, parentVertex);
        Edge ed = new Edge (x, y, parentVertex);
        ed.setLabelFontIndex (currentFont + 1);
        ed.setMetFontIndex (currentFont + 1);
        ex.setLabelFontIndex (currentFont + 1);
        ex.setMetFontIndex (currentFont + 1);
        ex.addOutEdge (ed);
        ed.setSource (ex);
        currentEdge = new EdgePath (ed);
        DisplayExternal extern = new DisplayExternal (ex);
        extern.setLabelShape ((Graphics2D) getGraphics ());
        extern.setMetShape ((Graphics2D) getGraphics ());      // ***
Maybe we don't need this ******
        displayComponentVector.addElement (extern);
        paintComponent (extern);
    }
    else {        // First click          // vertex-vertex or vertex-
external
        System.out.println ("Building edge" + v.getLabel ());
        IS_COLLECTING_POINTS = true;
        Edge ed = new Edge (x, y, parentVertex);
        ed.setLabelFontIndex (currentFont + 1);
        ed.setMetFontIndex (currentFont + 1);
        v.addOutEdge (ed);
        ed.setSource (v);
        currentEdge = new EdgePath (ed);
    }
}

/**
 * Paints the component into this panel by calling the
graphics2D.draw(Shape) method.
 *
 * @param component The component to be drawn into the panel
 */
```

```java
public void paintComponent (DisplayComponent component)
{
    Graphics2D g2D = (Graphics2D) getGraphics ();
    if (component instanceof DisplayVertex) {
        g2D.setColor (new Color (ColorConstants.RGB_VALUES
[currentColor]));
        //g2D.fill (((DisplayVertex) component).getPaintedShape ());
        g2D.fill (component.getShape ());
        g2D.setColor (Color.black);
    }
    g2D.draw (component.getShape ());
    component.drawLabelShape (g2D);
    if (component instanceof DisplayVertex)
        component.drawMetShape (g2D);
}

/**
 * This method is called to repaint all the components when necessary.
 *
 * @param g The graphics context of the panel
 */
public void paint (Graphics g)
{
    Graphics2D g2D = (Graphics2D) g;
    g2D.setColor (Color.black);      // Pending need to change the
color some
    //System.out.println ("Inside paint method of drawPanel" +
selectedComponent);

    // int size = displayComponentVector.size (); // *** WHY ? ******
    for (Enumeration e = displayComponentVector.elements ();
e.hasMoreElements ();) {
        DisplayComponent dcp = (DisplayComponent) e.nextElement ();
        DataFlowComponent dfc = dcp.getDataFlowComponent ();
        if (MOVING_COMPONENT ||              // *** PENDING Had to do
this to update in & out edges
            (MOVING_LABEL && selectedComponent.getDataFlowComponent ()
instanceof External))
            dcp.update ();        // and also label changes vs
        if (dfc instanceof Edge && ((Edge) dfc).isStateStream ()) {
            g2D.setStroke (new BasicStroke (1.5f));
            g2D.draw (dcp.getShape ());
            g2D.setStroke (new BasicStroke (1f));
        }
        g2D.setColor (Color.black);
        else {
            if (dcp instanceof DisplayVertex) {
                g2D.setColor (new Color (ColorConstants.RGB_VALUES
[((Vertex) dfc).getColor () - 1]));
                g2D.fill (((DisplayVertex) dcp).getPaintedShape ());
                g2D.setColor (Color.black);
            }
            g2D.draw (dcp.getShape ());

            if (!dfc.isLeaf ())
                g2D.draw (((DisplayVertex) dcp).getInnerShape ());
        }
        dcp.drawLabelShape (g2D);
        dcp.drawMetShape (g2D); // ***** Pending  **********
    }
    if ((selectMode && selectedComponent != null) || selectAllMode) {
        for (Enumeration e = handlesVector.elements (); e.hasMoreElements
();) {
            g2D.setColor (Color.gray);
            g2D.fill ((Shape) e.nextElement ());
        }
    }
}

/**
 * Sets the size of the panel to WIDTH and HEIGHT
 *
 * @return Returns a new Dimension object initialized to the
 *                WIDTH and HEIGHT parameters.
 */
public Dimension getPreferredSize ()
{
    return new Dimension (WIDTH, HEIGHT);
}

/**
 * Handles the event that occurs when a mouse button is clicked on
this panel
 *
 * @param e The MouseEvent that occurs.
 */
public void mousePressed (MouseEvent e)
{
    int xPosition = e.getX ();
    int yPosition = e.getY ();
    int flags = e.getModifiers ();
    prevPoint.setLocation (xPosition, yPosition);

    if (e.isAltDown ()) {
        // Do nothing for the middle Button at the moment
    }
    else if (flags == MouseEvent.BUTTON1_MASK || flags ==
MouseEvent.BUTTON3_MASK) {
        if (!selectAllMode && isHoldingHandle (xPosition, yPosition)) {
            System.out.println ("Holding Handle");
            if (selectedComponent instanceof DisplayVertex &&
                selectedComponent.getShape ().getBounds2D ().contains
(diagonalPoint)) // Make sure it is not the label
                RESIZING = true;
        }
```

224

```
    else if (selectMode) {
        //System.out.println ("Select mode is on");              // Just for
debugging
        DisplayComponent dc;
        boolean flag = false;
        for (Enumeration enum = displayComponentVector.elements ();
enum.hasMoreElements ();) {
            dc = (DisplayComponent) enum.nextElement ();
            if (selectAllMode && (dc.containsClickedPoint (xPosition,
yPosition) ||
                dc.getLabelShapeBounds ().contains (xPosition,
yPosition) ||
                (dc.getMetShapeBounds ().contains (xPosition,
yPosition)))) {
                System.out.println (selectedComponent + " Moving_All");
                MOVING_ALL = true;
                flag = true;
            }
            else if (dc.getLabelShapeBounds ().contains (xPosition,
yPosition) {    // If clicked on a label
                System.out.println ("Inside the label");
                eraseHandles ();              // ***** Pending this is too
simple
                handlesVector = dc.getStringHandles
(dc.getLabelShapeBounds ());
                selectedComponent = dc;
                paint (getGraphics ());
                flag = true;
            }
            else if (dc.getMetShapeBounds ().contains (xPosition,
yPosition) { // If clicked an an met
                System.out.println ("Inside the met");
                eraseHandles ();              // ***** Pending this is too
simple
                handlesVector = dc.getStringHandles
(dc.getMetShapeBounds ());
                selectedComponent = dc;
                paint (getGraphics ());
                flag = true;
            }
            // ***** PENDING *****
            else if (dc.containsClickedPoint (xPosition, yPosition)) {
// If clicked on a component
                System.out.println ("Yes, inside");              // This is more work can look at
                eraseHandles ();
it in the older versions
                handlesVector = dc.getHandles ();
                selectedComponent = dc;           // ***** Pending calls the
paint method twice here
                flag = true;  // Clicked on a component
            }
            else {

        if (flag)
            setCursor (MOVE_CURSOR);             // Clicked on an
    }
    if (selectedComponent != null && !flag) {    // I put it here
empty area
        selectedComponent = null;                // in the for loop
not to call  eraseHandles everytime
        eraseHandles ();
        setSelectAllMode (false);
    }
    setMenuBarItems ();
    //System.out.println (selectedComponent);
    }
    else if (flags != MouseEvent.BUTTON3_MASK) {
        parentVertex.setAllowsChildren (true);
        switch (currentComponent) {
        case OPERATOR:
            processOperator (xPosition, yPosition);
            if (selectionDefault)
                parentFrame.getToolBar ().enableSelectButton ();
                parentFrame.setSaveRequired (true);
            break;
        case TERMINATOR:
            processTerminator (xPosition, yPosition);
            if (selectionDefault)
                parentFrame.getToolBar ().enableSelectButton ();
                parentFrame.setSaveRequired (true);
            break;
        case STREAM:
            processStream (xPosition, yPosition, e.getClickCount
());  // Pending same as chriss' implementation
                parentFrame.setSaveRequired (true);
            break;
        default:
            break;
        }
    }
}

public void setMenuBarItems ()
{
    if (selectedComponent == null) {
        parentFrame.getMenuBar ().getMenu (1).getItem (4).setEnabled
(false);  // delete
        parentFrame.getMenuBar ().getMenu (3).getItem (2).setEnabled
(false);  // decompose
        parentFrame.getMenuBar ().getMenu (2).getItem (0).setEnabled
(true);   // color
    }
    else {
```

225

```java
(true); // delete
        parentFrame.getJMenuBar ().getMenu (1).getItem (4).setEnabled
(false); // color
        parentFrame.getJMenuBar ().getMenu (2).getItem (0).setEnabled
(false); // decompose
        parentFrame.getJMenuBar ().getMenu (3).getItem (2).setEnabled
        }
        else {
        parentFrame.getJMenuBar ().getMenu (2).getItem (0).setEnabled
(true); // color
        parentFrame.getJMenuBar ().getMenu (3).getItem (2).setEnabled
(true); // decompose
        }

    /**
     * Handles the event that occurs when the mouse enters into the panel.
     *
     * @param e The MouseEvent that occurs.
     */
    public void mouseEntered (MouseEvent e)
    {
        //System.out.println ("MouseEntered");
    }

    /**
     * Handles the event that occurs when the mouse exits the panel.
     *
     * @param e The MouseEvent that occurs.
     */
    public void mouseExited (MouseEvent e)
    {
        //System.out.println ("MouseExited");
    }

    /**
     * Handles the event that occurs when a mouse button is pressed on
this panel
     *
     * @param e The MouseEvent that occurs.
     */
    public void mouseClicked (MouseEvent e)
    {
        int xPosition = e.getX ();
        int yPosition = e.getY ();
        int flags = e.getModifiers ();
        int clickCount = e.getClickCount ();

        //System.out.println ("MouseClicked");

        if (selectedComponent != null &&
!(selectedComponent.getDataFlowComponent () instanceof External)
            && !selectAllMode) {
            if (flags == MouseEvent.BUTTON3_MASK)
                if (selectedComponent instanceof EdgePath)
                    popupMenu.showPopupMenu (true, xPosition, yPosition); //
disables decompose
                else
                    popupMenu.showPopupMenu (false, xPosition, yPosition); //
enables decompose
            else if (clickCount > 1)
                showProperties (selectedComponent);
        }
        else if (flags == MouseEvent.BUTTON3_MASK && IS_COLLECTING_POINTS) {
            External ex = new External (xPosition, yPosition, parentVertex);
            ex.addInEdge ((Edge) currentEdge.getDataFlowComponent ());
            ((Edge) currentEdge.getDataFlowComponent ()).setDestination (ex);
            ((Edge) currentEdge.getDataFlowComponent ()).addPoint (xPosition,
yPosition);
            parentFrame.getTreePanel ().addNewDFC ((Edge)
currentEdge.getDataFlowComponent (), parentVertex);
            currentEdge.setLabelShape ((Graphics2D) getGraphics ());
            currentEdge.setMetShape ((Graphics2D) getGraphics ());
            ((Edge) currentEdge.getDataFlowComponent ()).correctLabelOffset
();
            currentEdge.update ();
            IS_COLLECTING_POINTS = false;
            displayComponentVector.addElement (currentEdge);
            if (selectionDefault) {
                parentFrame.getToolBar ().enableSelectButton ();
                setSelectMode (true);
            }

            DisplayExternal extern = new DisplayExternal (ex);
            extern.setLabelShape ((Graphics2D) getGraphics ());
            extern.setMetShape ((Graphics2D) getGraphics ());       // ***
Maybe we don't need this ******
            displayComponentVector.addElement (extern);
            paintComponent (currentEdge);
            paintComponent (extern);
        }
    }

    public void showProperties (DisplayComponent d)
    {
        if (d instanceof DisplayVertex) {
            vPropertyPanel.setVertex ((Vertex) d.getDataFlowComponent ());
// *** Pending may also be stream ***
            vPropertyPanel.setDisplayVertex ((DisplayVertex) d);
        }
        else {
            ePropertyPanel.setEdge ((Edge) d.getDataFlowComponent ());
            ePropertyPanel.setEdgePath ((EdgePath) d);
```

```java
            }
        }

    /**
     * Handles the event that occurs when a mouse button is released on
this panel
     *
     * @param e The MouseEvent that occurs.
     */
    public void mouseReleased (MouseEvent e)
    {
        //System.out.println ("MouseReleased");
        MOVING_COMPONENT = false;
        MOVING_LABEL = false;
        MOVING_MET = false;
        RESIZING = false;
        MOVING_ALL = false;
        diagonalPoint = null;
        //setCursor (DEFAULT_CURSOR);
    }

    /**
     * Handles the event that occurs when the mouse is dragged on this
panel
     *
     * @param e The MouseEvent that occurs.
     */
    public void mouseDragged (MouseEvent e)
    {
        //System.out.println ("MouseDragged");
        if (selectedComponent != null || MOVING_ALL) {
            int xPosition = e.getX ();
            int yPosition = e.getY ();
            int flags = e.getModifiers ();

            // pending ????
            if (!MOVING_ALL) {
                bounds = (Rectangle) selectedComponent.getShape ().getBounds
();
            }
            System.out.println ("thisi isi it" + bounds);
            if (((bounds.getMinX () <= 0) && (xPosition <= prevPoint.x)) ||
                ((bounds.getMinY () <= 0) && (yPosition <= prevPoint.y)) ||
                ((bounds.getMaxX () >= WIDTH) && (xPosition >= prevPoint.x))
||
                ((bounds.getMaxY () >= HEIGHT) && (yPosition >=
prevPoint.y))) {
                setCursor (DEFAULT_CURSOR);
            }
            else if (flags == MouseEvent.BUTTON1_MASK) {
                DataFlowComponent dfc = selectedComponent.getDataFlowComponent
();

                if (selectAllMode && (MOVING_ALL)) {
                    //System.out.println ("Moving all");
                    handlesVector.removeAllElements ();
                    bounds = (Rectangle) ((DisplayComponent)
displayComponentVector.elementAt (0)
                        .getShape ()).getBounds ();
                    for (Enumeration enum = displayComponentVector.elements ();
enum.hasMoreElements ();) {
                        selectedComponent = (DisplayComponent) enum.nextElement
();

                        dfc = selectedComponent.getDataFlowComponent ();
                        dfc.moveTo (xPosition - prevPoint.x, yPosition -
prevPoint.y);

                        handlesVector.addAll (selectedComponent.getHandles ());
                        handlesVector.addAll (selectedComponent.getStringHandles
(selectedComponent.getLabelShapeBounds ()));
                        handlesVector.addAll (selectedComponent.getStringHandles
(selectedComponent.getMetShapeBounds ()));
                        selectedComponent.update ();
                        bounds = (Rectangle) bounds.createUnion
(selectedComponent.getShape ().getBounds ());
                    }
                }
                else if (RESIZING) {
                    System.out.println ("Now resizing");
                    Rectangle2D.Double r2D = (Rectangle2D.Double)
selectedComponent.getShape ().getBounds2D ();
                    r2D.setFrameFromDiagonal (new Point (xPosition, yPosition),
diagonalPoint);
                    ((Vertex) dfc).setX ((int) r2D.getCenterX ());
                    ((Vertex) dfc).setY ((int) r2D.getCenterY ());
                    ((Vertex) dfc).setWidth ((int) r2D.getWidth ());
                    selectedComponent.update ();
                    handlesVector = selectedComponent.getHandles ();
                }
                else if (MOVING_LABEL) {
                    if (dfc instanceof External) {
                        ((External) dfc).setLocation (xPosition - prevPoint.x,
yPosition - prevPoint.y);
                        selectedComponent.update ();
                    }
                    else
                        dfc.setLabelOffset (xPosition - prevPoint.x, yPosition -
prevPoint.y);
                    handlesVector = selectedComponent.getStringHandles
(selectedComponent.getLabelShapeBounds ());
                }
                else if (MOVING_MET) {
                    dfc.setMetOffset (xPosition - prevPoint.x, yPosition -
prevPoint.y);
                    handlesVector = selectedComponent.getStringHandles
(selectedComponent.getMetShapeBounds ());
                }
```

227

```java
            }
            else if (MOVING_COMPONENT) {
                if (dfc instanceof Vertex)
                    ((Vertex) dfc).setLocation (xPosition - prevPoint.x,
yPosition - prevPoint.y);
                else
                    ((Edge) dfc).reShape (xPosition, yPosition);
                selectedComponent.update ();
                handlesVector = selectedComponent.getHandles ();
            }
        }
        else {
            if (selectedComponent.getLabelShapeBounds ().contains
(xPosition, yPosition)) {
                MOVING_LABEL = true;
                if (dfc instanceof External) {
                    ((External) dfc).setLocation (xPosition -
prevPoint.y);
                    selectedComponent.update ();
                }
                else
                    dfc.setLabelOffset (xPosition - prevPoint.x,
yPosition - prevPoint.y);
                handlesVector = selectedComponent.getStringHandles
(selectedComponent.getLabelShapeBounds ());
                parentFrame.setSaveRequired (true);
            }
            if (selectedComponent.getMetShapeBounds ().contains
(xPosition, yPosition)) {
                MOVING_MET = true;
                dfc.setMetOffset (xPosition - prevPoint.x, yPosition -
prevPoint.y);
                handlesVector = selectedComponent.getStringHandles
(selectedComponent.getLabelShapeBounds ());
                parentFrame.setSaveRequired (true);
            }
            else if (selectedComponent.containsClickedPoint (xPosition,
yPosition)) {
                MOVING_COMPONENT = true;
                if (dfc instanceof Vertex)
                    ((Vertex) dfc).setLocation (xPosition - prevPoint.x,
yPosition - prevPoint.y);
                else
                    ((Edge) dfc).reShape (xPosition, yPosition);
                //selectedComponent.update ();
                handlesVector = selectedComponent.getHandles ();
                parentFrame.setSaveRequired (true);
            }
            //setCursor (MOVE_CURSOR);
        }
        clearAllComponentsFromScreen (null);
        paint (getGraphics ());
        setCursor (MOVE_CURSOR);


                prevPoint.setLocation (xPosition, yPosition);
            }
        }
    }

    public boolean isHoldingHandle (int x, int y)
    {
        boolean flag = false;
        Rectangle2D r2D;
        for (Enumeration e = handlesVector.elements (); e.hasMoreElements
();) {
            r2D = (Rectangle2D) e.nextElement ();
            if (r2D.contains (x, y)) {
                diagonalPoint = getDiagonalPoint (r2D);
                flag = true;
            }
        }
        return flag;
    }

    public Point2D getDiagonalPoint (Rectangle2D rect)
    {
        System.out.println ("Inside getDiagonalPoint");
        Point2D p = new Point2D.Double ();
        int w = (int) rect.getWidth () / 2;
        Rectangle2D r2D = (Rectangle2D) selectedComponent.getShape
().getBounds ();
        if (rect.getMaxX () >= r2D.getMaxX () && rect.getMaxY () >=
r2D.getMaxY ())
            p.setLocation (r2D.getMinX () + w, r2D.getMinY () + w);
        else if (rect.getMaxX () >= r2D.getMaxX () && rect.getMinY () <=
r2D.getMinY ())
            p.setLocation (r2D.getMinX () + w, r2D.getMaxY () - w);
        else if (rect.getMinX () <= r2D.getMinX () && rect.getMinY () <=
r2D.getMinY ())
            p.setLocation (r2D.getMaxX () - w, r2D.getMaxY () - w);
        else if (rect.getMinX () <= r2D.getMinX () && rect.getMaxY () >=
r2D.getMaxY ())
            p.setLocation (r2D.getMaxX () - w, r2D.getMinY () + w);
        System.out.println (p);
        return p;
    }

    /**
     * Handles the event that occurs when the mouse is moved on this panel
     *
     * @param e The MouseEvent that occurs.
     */
    public void mouseMoved (MouseEvent e)
    {
        //System.out.println ("Inside mouse moved");
        int xPosition = e.getX ();
```

```java
        int yPosition = e.getY ();
        if (selectMode) {
            setCursor (DEFAULT_CURSOR);
            DisplayComponent dc;
            for (Enumeration enum = displayComponentVector.elements ();
                 enum.hasMoreElements ();) {
                dc = (DisplayComponent) enum.nextElement ();
                if (dc.containsClickedPoint (xPosition, yPosition) ||
                    dc.getLabelShapeBounds ().contains (xPosition, yPosition)
                    || (dc.getMetShapeBounds ().contains (xPosition,
yPosition))) {
                    if (dc.equals (selectedComponent) && (MOVING_COMPONENT ||
MOVING_LABEL || MOVING_MET) )
                        setCursor (MOVE_CURSOR);
                    else
                        setCursor (HAND_CURSOR);
                }
            }
        }
//if (IS_COLLECTING_POINTS) {
//    rubberBandLine (prevPoint.x, prevPoint.y);
//    rubberBandLine (xPosition, yPosition);
//}

    public void actionPerformed (ActionEvent e)
    {
        if (e.getSource () == popupMenu.getDecomposeMenuItem ()) {
            decompose ();
        }
        if (e.getSource () == popupMenu.getFontMenuItem ()) {
            String selected = (String) JOptionPane.showInputDialog
(parentFrame, "Select Font : ",
                "Font Selection", JOptionPane.INFORMATION_MESSAGE,
null, FontConstants.FONT_NAMES,
                FontConstants.FONT_NAMES [0]);
            if (selected != null) {
                int fontIndex = 0;
                for (int ix = 0; ix < FontConstants.FONT_NAMES.length; ix++) {
                    if (FontConstants.FONT_NAMES [ix].equals (selected))
                        fontIndex = ix;
                }
                selectedComponent.getDataFlowComponent ().setLabelFontIndex
(fontIndex + 1);
                selectedComponent.getDataFlowComponent ().setMetFontIndex
(fontIndex + 1);
                selectedComponent.setLabelShape ((Graphics2D) getGraphics ());
                selectedComponent.setMetShape ((Graphics2D) getGraphics ());
                clearAllComponentsFromScreen ((Graphics2D) getGraphics ());
                paint (getGraphics ());
            }
        }
        else if (e.getSource () == popupMenu.getColorMenuItem ()) {
            String selected = (String) JOptionPane.showInputDialog
(parentFrame, "Select color : ",
                "Open", JOptionPane.INFORMATION_MESSAGE, null,
ColorConstants.COLOR_NAMES,
                ColorConstants.COLOR_NAMES [0]);
            if (selected != null) {
                int colorIndex = 0;
                for (int ix = 0; ix < ColorConstants.COLOR_NAMES.length; ix++)
                    if (ColorConstants.COLOR_NAMES [ix].equals (selected))
                        colorIndex = ix;
                selectedComponent.getDataFlowComponent ().setColor
(colorIndex + 1);
                ((Vertex) selectedComponent.getDataFlowComponent ((Graphics2D) getGraphics ());
                clearAllComponentsFromScreen ((Graphics2D) getGraphics ());
                paint (getGraphics ());
            }
        }
        else if (e.getSource () == popupMenu.getDeleteMenuItem ()) {
            deleteSelectedComponent ();
        }
        else if (e.getSource () == popupMenu.getPropMenuItem ()) {
            showProperties (selectedComponent);
        }
    }

    public void deleteSelectedComponent ()
    {
        if (!(selectedComponent.getDataFlowComponent () instanceof
External)) {
            selectedComponent.delete ();
            displayComponentVector.removeElement (selectedComponent);
            parentFrame.getTreePanel ().removeDfc
(selectedComponent.getDataFlowComponent ());          // This takes care
of unremoved streams
            setParentVertex (parentVertex, null);
            parentFrame.setSaveRequired (true);
        }
    }

    public Vertex getParentVertex ()
    {
        return parentVertex;
    }

    public void setSelectAllMode (boolean b)
    {
        selectAllMode = b;
        if (selectAllMode)
            selectAllComponents ();
```

```java
    }

    public void selectAllComponents ()
    {
        DisplayComponent dc = null;
        handlesVector.removeAllElements ();
        if (!displayComponentVector.isEmpty ()) {
            bounds = (Rectangle) ((DisplayComponent)
displayComponentVector.elementAt (0)).getShape ().getBounds ();
        }
        for (Enumeration enum = displayComponentVector.elements ();
enum.hasMoreElements ();) {
            dc = (DisplayComponent) enum.nextElement ();
            handlesVector.addAll (dc.getHandles ());
            handlesVector.addAll (dc.getStringHandles (dc.getLabelShapeBounds
()));
            handlesVector.addAll (dc.getStringHandles (dc.getMetShapeBounds
()));
            bounds = (Rectangle) bounds.createUnion (dc.getShape ().getBounds
());
        }
        selectedComponent = dc;
        paint (getGraphics ());
    }

    public void setSelectionDefault (boolean b)
    {
        selectionDefault = b;
    }

    public void setCurrentColor (int colorIndex)
    {
        currentColor = colorIndex;
    }

    public void setCurrentFont (int fontIndex)
    {
        currentFont = fontIndex;
    }

    protected void rubberBandLine (int x, int y)
    {
        Point last = (Point) ((Edge) currentEdge.getDataFlowComponent
()).getPoints ().lastElement ();
        Graphics g = getGraphics ();
        g.setColor (new Color (128, 128, 128));
        g.setXORMode (Color.white);
        g.drawLine (last.x, last.x, x, y);
        g.setPaintMode ();
        g.setColor (Color.black);
    }
} // End of the class DrawPanel.
```

230

```java
package caps.GraphEditor;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import caps.Psdl.*;
import caps.Display.EdgePath;
import caps.Parser.GrammarCheck;

public class EdgeProperties extends JDialog implements ActionListener {

    Edge targetEdge;

    EdgePath ePath;

    JTextField nameField;
    JTextField streamTypeField;
    JTextField latencyField;
    JTextField initValueField;

    JRadioButton noButton;
    JRadioButton yesButton;

    JComboBox latencyUnitsCombo;

    JButton okButton;
    JButton cancelButton;
    JButton helpButton;
    JButton initValueButton;

    Editor parentFrame;

    public EdgeProperties (Editor parent)
    {
        super (parent, "StreamProperties", true);
        parentFrame = parent;
        setResizable (false);
        initialize ();
        pack ();
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        setLocation ((screenSize.width - getWidth ()) / 2,
                     (screenSize.height - getHeight ()) / 2);
    }

    public void initialize ()
    {
        Box box = Box.createVerticalBox ();

        GridBagConstraints gbc = new GridBagConstraints ();
        gbc.fill = GridBagConstraints.BOTH;
        gbc.insets = new Insets (5, 3, 5, 3);

        JPanel namePanel = new JPanel (new GridBagLayout ());
        namePanel.setBorder (BorderFactory.createTitledBorder (""));
        nameField = new JTextField (15);
        streamTypeField = new JTextField (15);
        noButton = new JRadioButton ("No", true);
        noButton.addActionListener (this);
        yesButton = new JRadioButton ("Yes", false);
        yesButton.addActionListener (this);
        ButtonGroup group = new ButtonGroup ();
        group.add (noButton);
        group.add (yesButton);
        initValueButton = new JButton ("State Initial Value");
        initValueButton.addActionListener (this);
        initValueField = new JTextField (15);
        initValueField.setEditable (false);
        latencyField = new JTextField (10);
        latencyUnitsCombo = getUnitsCombo ();
        gbc.gridwidth = 2; gbc.gridx = 0; gbc.gridy = 0;
        namePanel.add (new JLabel ("Stream Name :"), gbc);
        gbc.gridwidth = 3; gbc.gridx = 2;
        namePanel.add (nameField, gbc);
        gbc.gridwidth = 2; gbc.gridx = 0; gbc.gridy = 1;
        namePanel.add (new JLabel ("Stream Type :"), gbc);
        gbc.gridwidth = 3; gbc.gridx = 2;
        namePanel.add (streamTypeField, gbc);
        gbc.gridwidth = 2; gbc.gridx = 0; gbc.gridy = 2;
        namePanel.add (new JLabel ("Is it a state stream ?"), gbc);
        gbc.gridwidth = 1; gbc.gridx = 2;
        namePanel.add (noButton, gbc);
        gbc.gridx = 3;
        namePanel.add (yesButton, gbc);
        gbc.gridwidth = 2; gbc.gridx = 0; gbc.gridy = 3;
        namePanel.add (initValueButton, gbc);
        gbc.gridwidth = 3; gbc.gridx = 2;
        namePanel.add (initValueField, gbc);
        gbc.gridwidth = 2; gbc.gridx = 0; gbc.gridy = 4;
        namePanel.add (new JLabel ("Latency :"), gbc);
        gbc.gridx = 2;
        namePanel.add (latencyField, gbc);
        gbc.gridwidth = 1; gbc.gridx = 4;
        namePanel.add (latencyUnitsCombo, gbc);

        JPanel okPanel = new JPanel (new FlowLayout ());
        okButton = new JButton ("OK");
        okButton.addActionListener (this);
        cancelButton = new JButton ("Cancel");
        cancelButton.addActionListener (this);
        helpButton = new JButton ("Help");
        helpButton.addActionListener (this);
        gbc.gridwidth = 1; gbc.gridx = 1; gbc.gridy = 0;
        okPanel.add (okButton, gbc);
        gbc.gridwidth = 1; gbc.gridx = 4;
```

```java
    okPanel.add (cancelButton, gbc);
    gbc.gridwidth = 1; gbc.gridx = 7;
    okPanel.add (helpButton, gbc);

    box.add (Box.createVerticalStrut (2));
    box.add (namePanel);
    box.add (Box.createVerticalStrut (5));
    box.add (okPanel);
    box.add (Box.createVerticalStrut (3));

    getContentPane ().add (box, BorderLayout.CENTER);
}

public void setEdge (Edge e)
{
    targetEdge = e;
    nameField.setText (e.getLabel ());
    streamTypeField.setText (e.getStreamType ());
    if (e.isStateStream () == false) {
        noButton.setSelected (true);
        initValueButton.setEnabled (false);
        initValueField.setText ("");
    }
    else {
        yesButton.setSelected (true);
        initValueButton.setEnabled (true);
        initValueField.setText (targetEdge.getInitialValue ());
    }
    if (e.getMet () != null) {
        PSDLTime latency = e.getMet ();
        latencyField.setText (String.valueOf (latency.getTimeValue ()));
        latencyUnitsCombo.setSelectedIndex (latency.getTimeUnits ());
    }
    else {
        latencyField.setText ("");
        latencyUnitsCombo.setSelectedIndex (1);    // Set default to ms
    }
}

public void setEdgePath (EdgePath e)
{
    ePath = e;
    setVisible (true);
}

public void actionPerformed (ActionEvent e)
{
    boolean exceptionOccurred = false;
    if (e.getSource () == yesButton) {
        initValueButton.setEnabled (true);
        initValueButton.doClick ();
    }
    if (e.getSource () == noButton) {
        initValueField.setText ("");
        initValueButton.setEnabled (false);
    }
    if (e.getSource () == initValueButton) {
        TextEditor.openDialog ("Stream Initial Value", "View or Edit
Stream Initial Value",                    initValueField.getText (),
GrammarCheck.INITIAL_EXPRESSION, false);
        initValueField.setText (TextEditor.getString ());
    }
    if (e.getSource () == okButton) {
        boolean errorStatus = false;
        String str = nameField.getText ();
        if (!GrammarCheck.isValid (str, GrammarCheck.ID)) {
            showErrorDialog ("Illegal stream name");
            errorStatus = true;
        }
        str = streamTypeField.getText ();
        if (!GrammarCheck.isValid (str, GrammarCheck.TYPE_NAME)) {
            showErrorDialog ("Illegal stream type name");
            errorStatus = true;
        }
        str = latencyField.getText ();
        if (str.length () != 0) {                  // To be able to delete
a latency value
            if (!GrammarCheck.isValid (str, GrammarCheck.INTEGER_LITERAL))
{
                showErrorDialog ("Illegal value for latency field");
                errorStatus = true;
            }
        }
        if (!errorStatus) {
            targetEdge.setLabel (nameField.getText ());
            targetEdge.setStreamType (streamTypeField.getText ());
            parentFrame.getDataTypes ().addType (streamTypeField.getText
());
            if (latencyField.getText ().length () != 0)
                targetEdge.setMet (new PSDLTime (Integer.parseInt
(latencyField.getText ()),                    latencyUnitsCombo.getSelectedIndex
()));
            else
                targetEdge.setMet (null);
            if (noButton.isSelected ()) {
                targetEdge.setInitialValue ("");
                targetEdge.setStateStream (false);
            }
            else {
                targetEdge.setInitialValue (initValueField.getText ());
```

232

```java
package caps.GraphEditor;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import caps.Psdl.Vertex;
import caps.Psdl.DataTypes;
import caps.Parser.*;
import java.io.*;

/**
 * The main frame for the Graph Editor.
 * It constructs and drives the other features.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class Editor extends JFrame implements Runnable(

/**
 * The panel that includes the Drawing area and tree view
 */
protected JPanel panel;

/**
 * Includes the treePanel and the drawPanel.
 */
protected JSplitPane innerSplit;

/**
 * The panel that includes the tree structure to view
 */
protected TreePanel treePanel;

/**
 * The panel that the drawing operations are performed.
 */
protected DrawPanel drawPanel;

protected StatusBar statusBar;

/**
 * the main toolbar of the GraphEditor
 */
protected ToolBar tBar;

protected Vertex root;

/**
 * The initial width of the GraphEditor
 */
```

```java
        targetEdge.setStateStream (true);
      }
      if (targetEdge.isStateStream ()) {
        Vertex parent = (Vertex) targetEdge.getParent ();
        ((java.util.Vector) parent.getSpecReqmts ().elementAt
(2)).addElement ("");
      }

      ePath.setLabelShape ((Graphics2D) parentFrame.getDrawPanel
().getGraphics ());
      ePath.setMetShape ((Graphics2D) parentFrame.getDrawPanel
().getGraphics ());
      setVisible (false);
      parentFrame.getDrawPanel ().clearAllComponentsFromScreen
(null);  // Is there a better way\
      parentFrame.getDrawPanel ().paint (parentFrame.getDrawPanel
().getGraphics ()); // Is there a better way\
      parentFrame.getTreePanel ().repaint ();
      parentFrame.setSaveRequired (true);

    }
    if (e.getSource () == cancelButton) {
      setVisible (false);

    }
    if (e.getSource () == helpButton) {
      System.out.println ("Help not available now");
    }

  }

  public void showErrorDialog (String str)
  {
    JOptionPane.showMessageDialog (this, str, "Error Message",
JOptionPane.ERROR_MESSAGE);
  }

  public JComboBox getUnitsCombo ()
  {
    JComboBox c =new JComboBox ();
    c.addItem ("microsec");
    c.addItem ("ms");
    c.addItem ("sec");
    c.addItem ("min");
    c.addItem ("hours");
    return c;

  }

} // End of the class EdgeProperties
```

```java
private final int INITIAL_WIDTH = 800;

/**
 * The initial height of the Graph Editor
 */
private final int INITIAL_HEIGHT = 600;

protected DataTypes types;

protected File prototypeFile;

protected boolean saveRequired;

/**
 * The constructor for the editor frame
 */
public Editor (File prototype, Vertex r, DataTypes t)
{
    //super ("PSDL EDitor"); // *** Must also show the title of the
prototype ***

    prototypeFile = prototype;

    root = r;

    saveRequired = false;

    types = t;

    //initialize ();
}

// this is another thread to paint main window
public void run ()
{
    initialize ();
}

/**
 * The initialization of the GUI takes place here
 */
public void initialize ()
{
    // Set the look and feel to a platform independent view
    try {
        UIManager.setLookAndFeel
(UIManager.getCrossPlatformLookAndFeelClassName ());
    } catch (Exception e) {
        System.err.println("Error loading L&F: " + e);
    }

    setTitle ("PSDL Editor : " + prototypeFile.getName ());

    setDefaultCloseOperation (WindowConstants.DO_NOTHING_ON_CLOSE);
    addWindowListener (new ExitEditor (this));

    statusBar = new StatusBar (this);

    /*
     * Construct the GUI here
     */
    setJMenuBar (new EditorMenuBar (this));

    setSize (INITIAL_WIDTH, INITIAL_HEIGHT);
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    setLocation((screenSize.width - INITIAL_WIDTH) / 2,
                (screenSize.height - INITIAL_HEIGHT) / 2);

    BorderLayout bLayout = new BorderLayout ();
    bLayout.setVgap (3);

    panel = new JPanel (bLayout);
    panel.setAlignmentX (LEFT_ALIGNMENT);
    panel.setAlignmentY (TOP_ALIGNMENT);
    panel.setBorder (BorderFactory.createLoweredBevelBorder ());

    getContentPane ().add (panel);

    tBar = new ToolBar (this);

    root.setAllowsChildren (true);

    treePanel = new TreePanel (this, root);
    drawPanel = new DrawPanel (this, root);
    JScrollPane p1 = new JScrollPane (treePanel);
    p1.setBackground (Color.white);
    JScrollPane p2 = new JScrollPane (drawPanel);
    p2.setBackground (Color.white);

    innerSplit = new JSplitPane (JSplitPane.HORIZONTAL_SPLIT, p1, p2);
    innerSplit.setContinuousLayout(true);
    innerSplit.setOneTouchExpandable (true);
    innerSplit.setDividerLocation (getWidth () / 5);
    innerSplit.setBorder (BorderFactory.createLoweredBevelBorder ());

    panel.add (tBar, BorderLayout.NORTH);
    panel.add (innerSplit, BorderLayout.CENTER);
    panel.add (statusBar, BorderLayout.SOUTH);

    PsdlParser.disable_tracing ();      // Set this to true if you want to
trace the parser actions

    setVisible (true);

    drawPanel.setParentVertex (root, null);
```

```
    return types;
}

public File getPrototypeFile ()
{
    return prototypeFile;
}

public void setSaveRequired (boolean b)
{
    saveRequired = b;
    if (saveRequired) {
        statusBar.setText ("Save required");
        getJMenuBar ().getMenu (0).getItem (0).setEnabled (true); // enable save menu item
    }
    else {
        statusBar.setText ("Save not required");
        getJMenuBar ().getMenu (0).getItem (0).setEnabled (false); // disable save menu item
    }
}

public boolean isSaveRequired ()
{
    return saveRequired;
}

public boolean checkSaved ()
{
    boolean value = true;
    if (saveRequired) {
        int ix = JOptionPane.showConfirmDialog (this, new String ("Save changes to the prototype?"));
        if (ix == JOptionPane.CANCEL_OPTION)
            value = false;
        else if (ix == JOptionPane.YES_OPTION)
            savePrototype ();
    }
    return value;
}

public void savePrototype ()
{
    try {
        setCursor (Cursor.WAIT_CURSOR);
        FileWriter testFile = new FileWriter (prototypeFile);
        StringWriter writer = new StringWriter ();
        CreatePsdl.ReInit (writer);
        CreatePsdl.build (root, types);
        String str = CreatePsdl.getPsdl ();
        testFile.write (str);
```

```
}

/**
 * *** Pending -- is it needed? ***
 * Returns the TreePanel object in this frame
 *
 * @return the treePanel object in this JFrame
 */
public TreePanel getTreePanel ()
{
    return treePanel;
}

/**
 * *** Pending -- is it needed? ***
 * Returns the DrawPanel object in this frame
 *
 * @return the drawPanel object in this JFrame
 */
public DrawPanel getDrawPanel ()
{
    return drawPanel;
}

/**
 * *** Pending -- is it needed? ***
 * Returns the toolBar object in this frame
 *
 * @return the toolBar object in this JFrame
 */
public ToolBar getToolBar ()
{
    return tBar;
}

public StatusBar getStatusBar ()
{
    return statusBar;
}

public JSplitPane getSplitPane ()
{
    return innerSplit;
}

public Vertex getRoot ()
{
    return root;
}

public DataTypes getDataTypes ()
{
```

```
            testFile.close ();
            PsdlParser.ReInit (new StringReader (str));
            PsdlParser.psdl ();
            writer.close ();
            setCursor (Cursor.DEFAULT_CURSOR);
            setSaveRequired (false);
        } catch (IOException ex) {
            System.out.println (ex);
        } catch (ParseException ex) {
            System.out.println (ex);
        //} catch (Exception ex) {
        //    System.out.println (ex);
        }
    }

} // End of the class Editor
```

```
package caps.GraphEditor;

import javax.swing.*;

/**
 * The MenuBar of the Graph Editor.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class EditorMenuBar extends JMenuBar {

    /**
     * The constructor for this class.
     */
    public EditorMenuBar (Editor parent)
    {
        super ();

        add (new GE_FileMenu (parent));
        add (new GE_EditMenu (parent));
        add (new GE_ViewMenu (parent));
        add (new GE_PSDLMenu (parent));
        add (new GE_HelpMenu (parent));
    }

} // End of the class EditorMenuBar
```

236

```
package caps.GraphEditor;

import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

/**
 * Closes the caps main window and exits from the program.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
class ExitEditor extends WindowAdapter {

    Editor editor;

    public ExitEditor (Editor e)
    {
        editor = e;
    }

    /**
     * Window event handler for the menu events.
     *
     * @param e The window event that is created when the program close
icon is pressed.
     */
    public void windowClosing (WindowEvent e)
    {
        if (editor.checkSaved ()) {
            caps.CAPSMain.CAPSMainWindow.removeEditor (editor);
            editor.setVisible (false);
            editor.dispose ();
        }
        //System.exit (0);    // *** Pending should normally close the window
only ***
    }

} // End of the class ExitEditor
```

```
package caps.GraphEditor;

public class FontConstants {

    public static String FONT_VALUES [] = { "Courier", "Bold", "10",
"Courier", "Bold", "12",
        "Courier", "Bold", "14", "Courier", "Plain", "10", "Courier",
"Plain", "12",
        "Courier", "Plain", "14" };

    public static String FONT_NAMES [] = { "Courier Bold 10", "Courier Bold
12", "Courier Bold 14",
        "Courier Plain 10", "Courier Plain 12", "Courier Plain 14" );

} // End of the class ColorConstants
```

```java
package caps.GraphEditor;

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

/**
 * Constructs the Edit menu of the menubar.
 * Also handles the events associated with the Edit Menu.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class GE_EditMenu extends JMenu implements ActionListener {

    /**
     * Initiates the 'Undo' event
     */
    private JMenuItem undoMenuItem = new JMenuItem ("Undo");

    /**
     * Initiates the 'Redo' event
     */
    private JMenuItem redoMenuItem = new JMenuItem ("Redo");

    /**
     * Initiates the 'Copy' event
     */
    //private JMenuItem copyMenuItem = new JMenuItem ("Copy");

    /**
     * Initiates the 'Paste' event
     */
    //private JMenuItem pasteMenuItem = new JMenuItem ("Paste");

    private JMenuItem selectAllMenuItem = new JMenuItem ("Select All");

    /**
     * Initiates the 'Delete' event
     */
    private JMenuItem deleteMenuItem = new JMenuItem ("Delete");

    private Editor parent;

    /**
     * The constructor for the Edit menu
     */
    public GE_EditMenu (Editor e)
    {
        super ("Edit");

        parent = e;

        add (undoMenuItem);
        add (redoMenuItem);
        addSeparator ();
        //add (copyMenuItem);
        //add (pasteMenuItem);
        add (selectAllMenuItem);
        add (deleteMenuItem);

        /*
         * These are not implemented yet
         * Take these lines out when they are implemented
         */
        undoMenuItem.setEnabled (false);
        redoMenuItem.setEnabled (false);
        deleteMenuItem.setEnabled (false);
        //copyMenuItem.setEnabled (false);
        //pasteMenuItem.setEnabled (false);

        undoMenuItem.setActionCommand ("Undo last action");
        redoMenuItem.setActionCommand ("Redo last action");
        //copyMenuItem.setActionCommand ("Copy selected component into
clipboard");
        //pasteMenuItem.setActionCommand ("Paste the component in the
clipboard into the drawing area");
        selectAllMenuItem.setActionCommand ("Selects all the components on
the drawing area");
        deleteMenuItem.setActionCommand ("Delete the selected component");

        undoMenuItem.addMouseListener (e.getStatusBar ());
        redoMenuItem.addMouseListener (e.getStatusBar ());
        selectAllMenuItem.addMouseListener (e.getStatusBar ());
        //copyMenuItem.addMouseListener (e.getStatusBar ());
        //pasteMenuItem.addMouseListener (e.getStatusBar ());
        deleteMenuItem.addMouseListener (e.getStatusBar ());

        undoMenuItem.addActionListener (this);
        redoMenuItem.addActionListener (this);
        selectAllMenuItem.addActionListener (this);
        //copyMenuItem.addActionListener (this);
        //pasteMenuItem.addActionListener (this);
        deleteMenuItem.addActionListener (this);
    }

    /**
     * Handles the menu events that occur when one of the menu items
     * is selected
     *
     * @param e The associated ActionEvent
     */
    public void actionPerformed (ActionEvent e)
    {
        if (e.getSource () == undoMenuItem) {
```

238

```
            System.out.println ("Undo has not yet been implemented");
        }
        else if (e.getSource () == redoMenuItem) {
            System.out.println ("Redo has not yet been implemented");
        }
//      else if (e.getSource () == copyMenuItem) {
//          System.out.println ("Copy has not yet been implemented");
//      }
//      else if (e.getSource () == pasteMenuItem) {
//          System.out.println ("Paste has not yet been implemented");
//      }
        else if (e.getSource () == selectAllMenuItem) {
            parent.getDrawPanel ().setSelectAllMode (true);
        }
        else if (e.getSource () == deleteMenuItem) {
            parent.getDrawPanel ().deleteSelectedComponent ();
        }
    }

} // End of the class GE_EditMenu
```

```
package caps.GraphEditor;

import javax.swing.*;
import java.awt.event.*;
import java.awt.Cursor;
import caps.Psdl.Vertex;

/**
 * Constructs the File menu of the menubar.
 * Also handles the events associated with the File Menu.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class GE_FileMenu extends JMenu implements ActionListener {

    /**
     * Initiates the 'Save' event
     */
    private JMenuItem saveMenuItem = new JMenuItem ("Save");

    /**
     * Initiates the 'Restore From Save' event
     */
    private JMenuItem restoreMenuItem = new JMenuItem ("Restore From Save");

    /**
     * Initiates the 'Print' event
     */
    private JMenuItem printMenuItem = new JMenuItem ("Print");

    /**
     * Initiates the 'Exit' event
     */
    private JMenuItem exitMenuItem = new JMenuItem ("Exit");

    private Editor parent;

    /**
     * The constructor for the File menu
     */
    public GE_FileMenu (Editor e)
    {
        super ("File");

        parent = e;

        add (saveMenuItem);
        add (restoreMenuItem);
        add (printMenuItem);
        add (exitMenuItem);
```

```
    /*
       These are not implemented yet
       Take these lines out when they are implemented
    */
    restoreMenuItem.setEnabled (false);

    saveMenuItem.setEnabled (false);

    saveMenuItem.setActionCommand ("Save the prototype into disk");
    restoreMenuItem.setActionCommand ("Restore saved prototype from
disk");

    printMenuItem.setActionCommand ("Print the prototype");
    exitMenuItem.setActionCommand ("Quit the graph editor");

    saveMenuItem.addMouseListener (e.getStatusBar ());
    restoreMenuItem.addMouseListener (e.getStatusBar ());
    printMenuItem.addMouseListener (e.getStatusBar ());
    exitMenuItem.addMouseListener (e.getStatusBar ());

    saveMenuItem.addActionListener (this);
    restoreMenuItem.addActionListener (this);
    printMenuItem.addActionListener (this);
    exitMenuItem.addActionListener (this);
}

/**
 * Handles the menu events that occur when one of the menu items
 * is selected
 *
 * @param e The associated ActionEvent
 */
public void actionPerformed (ActionEvent e)
{
    if (e.getSource () == saveMenuItem) {
        parent.savePrototype ();
    }
    else if (e.getSource () == restoreMenuItem) {
        System.out.println ("Restore has not yet been implemented");
    }
    else if (e.getSource () == printMenuItem) {
        DrawPanel panel = parent.getDrawPanel ();
        PrintJob.print (panel, (Vertex) panel.getParentVertex ().getRoot
());
    }
    else if (e.getSource () == exitMenuItem) {
        if (parent.checkSaved ()) {
            caps.CAPSMain.CAPSMainWindow.removeEditor (parent);
            parent.dispose ();
            parent.setVisible (false);
        }
    }
}

} // End of the class GE_FileMenu
```

240

```java
package caps.GraphEditor;

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

/**
 * Constructs the Help menu of the menubar.
 * Also handles the events associated with the Help Menu.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class GE_HelpMenu extends JMenu implements ActionListener {

    /**
     * Initiates the 'PSDL Grammar' event
     */
    private JMenuItem psdlGrammarMenuItem = new JMenuItem ("PSDL Grammar");

    /**
     * Initiates the 'Operators' event
     */
    private JMenuItem operatorsMenuItem = new JMenuItem ("Operators");

    /**
     * Initiates the 'Streams' event
     */
    private JMenuItem streamsMenuItem = new JMenuItem ("Streams");

    /**
     * Initiates the 'Exceptions' event
     */
    private JMenuItem exceptionsMenuItem = new JMenuItem ("Exceptions");

    /**
     * Initiates the 'Timers' event
     */
    private JMenuItem timersMenuItem = new JMenuItem ("Timers");

    /**
     * The constructor for the Help menu
     */
    public GE_HelpMenu (Editor e)
    {
        super ("Help");

        add (psdlGrammarMenuItem);
        add (operatorsMenuItem);
        add (streamsMenuItem);
        add (exceptionsMenuItem);
        add (timersMenuItem);

        /*
         * These are not implemented yet
         * Take these lines out when they are implemented
         */
        psdlGrammarMenuItem.setEnabled (false);
        operatorsMenuItem.setEnabled (false);
        streamsMenuItem.setEnabled (false);
        exceptionsMenuItem.setEnabled (false);
        timersMenuItem.setEnabled (false);

        psdlGrammarMenuItem.setActionCommand ("Opens help about PSDL Grammar");
        operatorsMenuItem.setActionCommand ("Opens help about operators");
        streamsMenuItem.setActionCommand ("Opens help about streams");
        exceptionsMenuItem.setActionCommand ("Opens help about exceptions");
        timersMenuItem.setActionCommand ("Opens help about timers");

        psdlGrammarMenuItem.addMouseListener (e.getStatusBar ());
        operatorsMenuItem.addMouseListener (e.getStatusBar ());
        streamsMenuItem.addMouseListener (e.getStatusBar ());
        exceptionsMenuItem.addMouseListener (e.getStatusBar ());
        timersMenuItem.addMouseListener (e.getStatusBar ());

        psdlGrammarMenuItem.addActionListener (this);
        operatorsMenuItem.addActionListener (this);
        streamsMenuItem.addActionListener (this);
        exceptionsMenuItem.addActionListener (this);
        timersMenuItem.addActionListener (this);
    }

    /**
     * Handles the menu events that occur when one of the menu items
     * is selected
     *
     * @param e The associated ActionEvent
     */
    public void actionPerformed (ActionEvent e)
    {
        if (e.getSource () == psdlGrammarMenuItem) {
            System.out.println ("PSDL Grammar help has not yet been implemented");
        }
        else if (e.getSource () == operatorsMenuItem) {
            System.out.println ("Operators help has not yet been implemented");
        }
        else if (e.getSource () == streamsMenuItem) {
            System.out.println ("Streams help has not yet been implemented");
        }
        else if (e.getSource () == exceptionsMenuItem) {
            System.out.println ("Exceptions help has not yet been implemented");
```

```java
        }
        else if (e.getSource () == timersMenuItem) {
            System.out.println ("Timers help has not yet been implemented");
        }
    }

} // End of the class GE_HelpMenu
```

```java
package caps.GraphEditor;

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

/**
 * Constructs the PSDL menu of the menubar.
 * Also handles the events associated with the PSDL Menu.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class GE_PSDLMenu extends JMenu implements ActionListener {

    /**
     * Initiates the 'Goto Root' event
     */
    private JMenuItem gotoRootMenuItem = new JMenuItem ("Goto Root");

    /**
     * Initiates the 'Goto Parent' event
     */
    private JMenuItem gotoParentMenuItem = new JMenuItem ("Goto Parent");

    /**
     * Initiates the 'Decompose' event
     */
    private JMenuItem decomposeMenuItem = new JMenuItem ("Decompose");

    private Editor parent;

    /**
     * The constructor for the PSDL menu
     */
    public GE_PSDLMenu (Editor e)
    {
        super ("PSDL");

        parent = e;

        add (gotoRootMenuItem);
        add (gotoParentMenuItem);
        add (decomposeMenuItem);

        decomposeMenuItem.setEnabled (false);
        gotoParentMenuItem.setEnabled (false);

        gotoRootMenuItem.setActionCommand ("Goto the root operator");
        gotoParentMenuItem.setActionCommand ("Goto the parent vertex");
        decomposeMenuItem.setActionCommand ("Decompose the selected
component");
```

242

```java
package caps.GraphEditor;

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

/**
 * Constructs the View menu of the menubar.
 * Also handles the events associated with the View Menu.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class GE_ViewMenu extends JMenu implements ActionListener {

  /**
   * Initiates the 'Color' event
   */
  private JMenuItem colorMenuItem = new JMenuItem ("Color");

  /**
   * Initiates the 'Font' event
   */
  private JMenuItem fontMenuItem = new JMenuItem ("Font");

  /**
   * Initiates the 'Refresh' event
   */
  private JMenuItem refreshMenuItem = new JMenuItem ("Refresh");

  /**
   * Initiates the 'Tree View' event
   */
  private JCheckBoxMenuItem treeViewMenuItem = new JCheckBoxMenuItem
("Tree");

  private JCheckBoxMenuItem toolTipsMenuItem = new JCheckBoxMenuItem
("Tool Tips");

  private JCheckBoxMenuItem selectionModeMenuItem = new JCheckBoxMenuItem
("Auto Select Mode");

  private ToolTipManager manager;

  private Editor parentFrame;

  /**
   * The constructor for the View menu
   */
  public GE_ViewMenu (Editor parent)
  {
    super ("View");

    gotoRootMenuItem.addMouseListener (parent.getStatusBar ());
    gotoParentMenuItem.addMouseListener (parent.getStatusBar ());
    decomposeMenuItem.addMouseListener (parent.getStatusBar ());

    gotoRootMenuItem.addActionListener (this);
    gotoParentMenuItem.addActionListener (this);
    decomposeMenuItem.addActionListener (this);
  }

  /**
   * Handles the menu events that occur when one of the menu items
   * is selected
   *
   * @param e The associated ActionEvent
   */
  public void actionPerformed (ActionEvent e)
  {
    if (e.getSource () == gotoRootMenuItem) {
      parent.getDrawPanel ().gotoRoot ();
    }
    else if (e.getSource () == gotoParentMenuItem) {
      parent.getDrawPanel ().gotoParent ();
    }
    else if (e.getSource () == decomposeMenuItem) {
      parent.getDrawPanel ().decompose ();
    }
  }

} // End of the class GE_PSDLMenu
```

```java
parentFrame = parent;

treeViewMenuItem.setSelected (true);
toolTipsMenuItem.setSelected (true);
selectionModeMenuItem.setSelected (false);

manager = ToolTipManager.sharedInstance ();
manager.setEnabled (true);
manager.setInitialDelay (400);

add (colorMenuItem);
add (fontMenuItem);
addSeparator ();
add (treeViewMenuItem);
add (toolTipsMenuItem);
add (selectionModeMenuItem);
addSeparator ();
add (refreshMenuItem);

/*
    These are not implemented yet
    Take these lines out when they are implemented
*/
//colorMenuItem.setEnabled (false);
//fontMenuItem.setEnabled (false);

colorMenuItem.setActionCommand ("Changes the current color of the
editor");
fontMenuItem.setActionCommand ("Changes the current font of the
editor");
treeViewMenuItem.setActionCommand ("Makes visible/hides the tree
view");
toolTipsMenuItem.setActionCommand ("Enables/Disables tooltips");
refreshMenuItem.setActionCommand ("Refreshes the display on the
drawing area");
selectionModeMenuItem.setActionCommand ("Defaults to select mode
                           " on the drawing area");
after each insertion of a component" +

colorMenuItem.addMouseListener (parentFrame.getStatusBar ());
fontMenuItem.addMouseListener (parentFrame.getStatusBar ());
treeViewMenuItem.addMouseListener (parentFrame.getStatusBar ());
toolTipsMenuItem.addMouseListener (parentFrame.getStatusBar ());
refreshMenuItem.addMouseListener (parentFrame.getStatusBar ());
selectionModeMenuItem.addMouseListener (parentFrame.getStatusBar
());

colorMenuItem.addActionListener (this);
fontMenuItem.addActionListener (this);
treeViewMenuItem.addActionListener (this);
toolTipsMenuItem.addActionListener (this);
refreshMenuItem.addActionListener (this);

selectionModeMenuItem.addActionListener (this);
}

/**
 * Handles the menu events that occur when one of the menu items
 * is selected
 *
 * @param e The associated ActionEvent
 */
public void actionPerformed (ActionEvent e)
{
    if (e.getSource () == colorMenuItem) {
        String selected = (String) JOptionPane.showInputDialog
(parentFrame, "Select color : ",
                            "Color Selection",
JOptionPane.INFORMATION_MESSAGE, null, ColorConstants.COLOR_NAMES,
                            ColorConstants.COLOR_NAMES [0]);
        if (selected != null) {
            int colorIndex = 0;
            for (int ix = 0; ix < ColorConstants.COLOR_NAMES.length; ix++)
                if (ColorConstants.COLOR_NAMES [ix].equals (selected))
                    colorIndex = ix;
            parentFrame.getDrawPanel ().setCurrentColor (colorIndex);
        }
    }
    else if (e.getSource () == fontMenuItem) {
        String selected = (String) JOptionPane.showInputDialog
(parentFrame, "Select Font : ",
                            "Font Selection", JOptionPane.INFORMATION_MESSAGE,
null, FontConstants.FONT_NAMES,
                            FontConstants.FONT_NAMES [0]);
        if (selected != null) {
            int fontIndex = 0;
            for (int ix = 0; ix < FontConstants.FONT_NAMES.length; ix++) {
                if (FontConstants.FONT_NAMES [ix].equals (selected))
                    fontIndex = ix;
            parentFrame.getDrawPanel ().setCurrentFont (fontIndex);
        }
    }
    else if (e.getSource () == treeViewMenuItem) {
        if (!treeViewMenuItem.isSelected ())
            parentFrame.getSplitPane ().setDividerLocation (0.0);
        else
            parentFrame.getSplitPane ().setDividerLocation (
                    parentFrame.getSplitPane ().getLastDividerLocation
());
    }
    else if (e.getSource () == toolTipsMenuItem) {
        if (toolTipsMenuItem.isSelected ())
```

```java
        manager.setEnabled (true);
    else
        manager.setEnabled (false);
    }
    else if (e.getSource () == refreshMenuItem) {
        DrawPanel panel = parentFrame.getDrawPanel ();
        panel.paint (panel.getGraphics ());
    }
    else if (e.getSource () == selectionModeMenuItem) {
        if (selectionModeMenuItem.isSelected ())
            parentFrame.getDrawPanel ().setSelectionDefault (true);
        else
            parentFrame.getDrawPanel ().setSelectionDefault (false);
    }

} // End of the class GE_ViewMenu


package caps.GraphEditor;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import caps.Parser.GrammarCheck;
import java.util.*;

public class IdListEditor implements ActionListener {

    private static JDialog dialog;

    private static JPanel south;

    private static final int WIDTH = 400;

    private static final int HEIGHT = 300;

    protected static Vector vector;

    protected static JButton okButton;
    protected static JButton cancelButton;
    protected static JButton helpButton;
    protected static JButton addButton;
    protected static JButton deleteButton;
    protected static JButton editButton;

    protected static JList inputArea;

    protected static DefaultListModel model;

    protected static JLabel promptLabel;

    protected Editor parentFrame;

    public IdListEditor (Editor parent)
    {
        parentFrame = parent;
        dialog = new JDialog (parentFrame, true);
        dialog.setSize (WIDTH, HEIGHT);
        dialog.setResizable (false);
        dialog.setVisible (false);
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        dialog.setLocation ((screenSize.width - dialog.getWidth ()) / 2,
                            (screenSize.height - dialog.getHeight ()) / 2);
        initialize ();
    }

    protected void initialize ()
    {
        dialog.getContentPane ().setLayout (new BorderLayout(5, 5));
```

```java
        south = new JPanel ();
        dialog.getContentPane ().add (south, BorderLayout.SOUTH);

        promptLabel = new JLabel ();
        dialog.getContentPane ().add (promptLabel, BorderLayout.NORTH);

        model = new DefaultListModel ();
        inputArea = new JList (model);
        inputArea.setBorder (BorderFactory.createLoweredBevelBorder ());
        JScrollPane p = new JScrollPane (inputArea);
        p.setBackground (Color.lightGray);
        dialog.getContentPane ().add (p, BorderLayout.CENTER);

        okButton = new JButton ("OK");
        okButton.addActionListener (this);
        cancelButton = new JButton ("Cancel");
        cancelButton.addActionListener (this);
        helpButton = new JButton ("Help");
        helpButton.addActionListener (this);
        addButton = new JButton ("Add");
        addButton.addActionListener (this);
        deleteButton = new JButton ("Delete");
        deleteButton.addActionListener (this);
        editButton = new JButton ("Edit");
        editButton.addActionListener (this);

        dialog.setTitle ("ID List");
        promptLabel.setText ("Enter or Edit IDs");

        south.add (okButton);
        south.add (cancelButton);
        south.add (addButton);
        south.add (deleteButton);
        south.add (editButton);
        south.add (helpButton);
    }

    public static void openDialog (Vector v)
    {
        vector = (Vector) v.clone ();
        setListElements ();
        inputArea.requestFocus ();
        dialog.setVisible (true);
    }

    public static void setListElements ()
    {
        model.removeAllElements ();
        for (Enumeration enum = vector.elements (); enum.hasMoreElements
            ();) {
            model.addElement (enum.nextElement ());
        }
    }

    public static Vector getIDList ()
    {
        return (Vector) vector.clone ();
    }

    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource () == okButton) {
            vector.removeAllElements ();
            for (Enumeration enum = model.elements (); enum.hasMoreElements
                ();) {
                vector.addElement (enum.nextElement ());
            }
            dialog.setVisible(false);
            parentFrame.setSaveRequired (true);
        }
        else if (e.getSource () == cancelButton) {
            dialog.setVisible(false);
        }
        else if (e.getSource () == addButton) {
            String newId = showInputDialog ();
            if (newId != null && newId.length () != 0) {
                if (GrammarCheck.isValid (newId, GrammarCheck.ID)) {
                    model.addElement (newId);
                    inputArea.setSelectedValue (newId, true);
                }
                else
                    showErrorDialog ("Illegal id value entered");
            }
        }
        else if (e.getSource () == deleteButton) {
            int index = inputArea.getSelectedIndex ();
            if (index >= 0)                // If there is a selected elements
                model.removeElementAt (index);
            else
                showErrorDialog ("Please select an Id to delete");
        }
        else if (e.getSource () == editButton) {
            int index = inputArea.getSelectedIndex ();
            if (index >= 0) {
                String editedId = showEditDialog ((String)
                    inputArea.getSelectedValue ());
                if (editedId != null && editedId.length () != 0) {
                    if (GrammarCheck.isValid (editedId, GrammarCheck.ID))
                        model.setElementAt (editedId, index);
                    else
                        showErrorDialog ("Illegal Id value enetered");
                }
                else if (editedId.length () == 0)
                    model.removeElementAt (index)
```

246

```java
        }
        else
            showErrorDialog ("Please select an Id to edit");
    }
    else if (e.getSource () == helpButton) {

    }

    public void showErrorDialog (String str)
    {
        JOptionPane.showMessageDialog (dialog, str, "Error Message",
JOptionPane.ERROR_MESSAGE);
    }

    public String showInputDialog ()
    {
        return JOptionPane.showInputDialog (dialog, "Enter new Id", "ID
Input Dialog", JOptionPane.QUESTION_MESSAGE);
    }

    public String showEditDialog (String initial)
    {
        return (String) JOptionPane.showInputDialog (dialog, "Edit Id", "ID
Edit Dialog",
JOptionPane.QUESTION_MESSAGE, null, null, initial);
    }
}
```

```java
package caps.GraphEditor;

import javax.swing.JMenuItem;
import javax.swing.JPopupMenu;

public class Popup extends JPopupMenu {

JMenuItem decomposeMenuItem = new JMenuItem ("Decompose");

JMenuItem fontMenuItem = new JMenuItem ("Font");

JMenuItem colorMenuItem = new JMenuItem ("Color");

JMenuItem deleteMenuItem = new JMenuItem ("Delete");

JMenuItem propMenuItem = new JMenuItem ("Properties");

DrawPanel panel;

public Popup (DrawPanel parent)
{
    super ();

    panel = parent;

    add (propMenuItem);
    addSeparator ();
    add (decomposeMenuItem);
    addSeparator ();
    add (deleteMenuItem);
    addSeparator ();
    add (fontMenuItem);
    add (colorMenuItem);

    decomposeMenuItem.addActionListener (parent);
    fontMenuItem.addActionListener (parent);
    colorMenuItem.addActionListener (parent);
    deleteMenuItem.addActionListener (parent);
    propMenuItem.addActionListener (parent);
}

public JMenuItem getDecomposeMenuItem ()
{
    return decomposeMenuItem;
}

public JMenuItem getFontMenuItem ()
{
    return fontMenuItem;
}

public JMenuItem getColorMenuItem ()
```

```
    return colorMenuItem;
}

public JMenuItem getDeleteMenuItem ()
{
    return deleteMenuItem;
}

public JMenuItem getPropMenuItem ()
{
    return propMenuItem;
}

public void showPopupMenu (boolean isEdge, int x, int y)
{
    if (isEdge) {
        decomposeMenuItem.setEnabled (false);
        colorMenuItem.setEnabled (false);
    }
    else {
        decomposeMenuItem.setEnabled (true);
        colorMenuItem.setEnabled (true);
    }
    pack ();
    show (panel, x, y);
}

} // End of the class Popup
```

```
package caps.GraphEditor;

import java.awt.print.*;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.BasicStroke;
import java.awt.geom.*;
import caps.Psdl.*;
import java.util.Enumeration;
import java.util.Vector;

public class PrintJob implements Runnable, Printable, Pageable {

    Vector printablePages;

    PrinterJob printJob;

    PageFormat format;

    DrawPanel panel;

    int orientation;

    public PrintJob (DrawPanel p, Vertex root)
    {
        panel = p;

        printablePages = new Vector (0, 2);

        orientation = PageFormat.PORTRAIT;

        DataFlowComponent d;
        for (Enumeration enum = root.breadthFirstEnumeration ();
        enum.hasMoreElements ();) {
            d = (DataFlowComponent) enum.nextElement ();
            if (d instanceof Vertex && !d.isLeaf ())
                printablePages.addElement ((Vertex) d);
        }
    }

    public void run ()
    {
        printJob = PrinterJob.getPrinterJob();
        printJob.setPageable (this);
        PageFormat f = printJob.defaultPage ();
        try {
            format = printJob.defaultPage ();
            format = printJob.pageDialog (f);
            if (!f.equals (format)) {       //if cancel is not selected
                printJob.print();
            }
        } catch (Exception ex) {
```

```java
        ex.printStackTrace();
    }
}

public static void print (DrawPanel p, Vertex root)
{
    PrintJob newJob = new PrintJob (p, root);
    new Thread (newJob).start ();
}

public int print (Graphics g, PageFormat f, int pi)
{
    if (pi >= printablePages.size ()) {
        return Printable.NO_SUCH_PAGE;
    }

    Graphics2D g2D = (Graphics2D) g;

    double scale = f.getImageableWidth () / (DrawPanel.WIDTH + 20);
    if (orientation == PageFormat.PORTRAIT) {
        g2D.translate ((f.getWidth () - DrawPanel.WIDTH * scale) / 2,
                       (f.getHeight () - DrawPanel.HEIGHT * scale) / 2);
    }
    else {
        scale = f.getImageableWidth () / (double) (DrawPanel.HEIGHT + 35
+ 10 + 10);
        g2D.translate (((DrawPanel.WIDTH * scale - f.getWidth () / 2 +
f.getImageableWidth () +
                        f.getImageableX () - (25 * scale)),
                       (f.getImageableY () + (20 * scale)));
        g2D.rotate (Math.toRadians (90));
    }

    g2D.scale (scale, scale);

    panel.setParentVertex ((Vertex) printablePages.elementAt (pi), g2D);
    // Draws components into graphics device

    g2D.setStroke (new BasicStroke (1.5f));
    g2D.draw (new Rectangle2D.Double (-5, -5, 1029, 773));      // bounding
rectangle around the prototype
    g2D.setStroke (new BasicStroke (1f));
    g2D.drawString ("Parent Vertex : " + ((Vertex)
printablePages.elementAt (pi)).getLabel (), 0, -30);

    return Printable.PAGE_EXISTS;
}

public int getNumberOfPages ()
{
    return printablePages.size ();
}

public PageFormat getPageFormat (int pageIndex)
{
    if (format.getOrientation () == PageFormat.LANDSCAPE) {
        orientation = PageFormat.LANDSCAPE;
        Paper p = new Paper ();
        format.setOrientation (PageFormat.PORTRAIT);
        format.setPaper (p);
    }
    else {
        Paper p = new Paper ();          // Disable margin settings
        format.setPaper (p);             // They will have no effect
    }
    return format;
}

public Printable getPrintable (int pageIndex)
{
    return this;
}

}  // End of the class PrintJob
```

```java
package caps.GraphEditor;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import caps.Parser.GrammarCheck;

public class TextEditor implements ActionListener {

    private static JDialog dialog;

    private static JPanel south;

    private static final int WIDTH = 400;

    private static final int HEIGHT = 300;

    private static int grammarKind;

    protected static JButton okButton;
    protected static JButton cancelButton;
    protected static JButton helpButton;

    protected static JTextArea inputArea;

    protected static JLabel promptLabel;

    static boolean allowsEmptyString;

    static String text;

    protected Editor parentFrame;

    public TextEditor (Editor parent)
    {
        parentFrame = parent;
        dialog = new JDialog (parentFrame, true);
        dialog.setSize (WIDTH, HEIGHT);
        dialog.setResizable (false);
        dialog.setVisible (false);
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        dialog.setLocation ((screenSize.width - dialog.getWidth ()) / 2,
                            (screenSize.height - dialog.getHeight ()) / 2);
        initialize ();
    }

    protected void initialize ()
    {
        dialog.getContentPane ().setLayout (new BorderLayout(5, 5));

        south = new JPanel ();
        dialog.getContentPane ().add (south, BorderLayout.SOUTH);
```

```java
package caps.GraphEditor;

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

public class StatusBar extends JLabel implements MouseListener {

    Editor parent;

    public StatusBar (Editor e)
    {
        super ("Save not required");

        parent = e;
    }

    public void mouseEntered (MouseEvent e)
    {
        setText (((AbstractButton) e.getSource ()).getActionCommand ());
    }

    public void mouseExited (MouseEvent e)
    {
        if (parent.isSaveRequired ())
            setText ("Save required");
        else
            setText ("Save not required");
    }

    public void mouseClicked (MouseEvent e)
    {
    }

    public void mousePressed (MouseEvent e)
    {
    }

    public void mouseDragged (MouseEvent e)
    {
    }

    public void mouseReleased (MouseEvent e)
    {
    }

} // End of the class StatusBar
```

```
            else {
                showErrorDialog ("Illegal value entered");
                errorStatus = true;
            }
        }
        else {
            text = "";
        }
        if (!errorStatus) {
            dialog.setVisible(false);
            parentFrame.setSaveRequired (true);
        }
    }
    else if (e.getSource () == cancelButton) {
        dialog.setVisible(false);
    }
    else if (e.getSource () == helpButton) {
    }
}

public void showErrorDialog (String str)
{
    JOptionPane.showMessageDialog (dialog, str, "Error Message",
JOptionPane.ERROR_MESSAGE);
}
```

```
        promptLabel = new JLabel ();
        dialog.getContentPane ().add (promptLabel, BorderLayout.NORTH);

        inputArea = new JTextArea ();
        inputArea.setLineWrap (true);
        inputArea.setBorder (BorderFactory.createLoweredBevelBorder ());
        JScrollPane p = new JScrollPane (inputArea);
        p.setBackground (Color.lightGray);
        dialog.getContentPane ().add (p, BorderLayout.CENTER);

        okButton = new JButton ("OK");
        okButton.addActionListener (this);

        cancelButton = new JButton ("Cancel");
        cancelButton.addActionListener (this);

        helpButton = new JButton ("Help");
        helpButton.addActionListener (this);

        south.add (okButton);
        south.add (cancelButton);
        south.add (helpButton);
    }

    public static void openDialog (String title, String prompt, String str,
int kind, boolean flag)
    {
        dialog.setTitle (title);
        promptLabel.setText (prompt);
        text = str;
        inputArea.setText (str);
        inputArea.requestFocus ();
        grammarKind = kind;
        allowsEmptyString = flag;
        dialog.setVisible (true);
    }

    public static String getString ()
    {
        text = text.trim ();
        return text;
    }

    public void actionPerformed (ActionEvent e)
    {
        if (e.getSource () == okButton) {
            boolean errorStatus = false;
            String str = inputArea.getText ();
            if (str.length () != 0 || !allowsEmptyString) {
                if (GrammarCheck.isValid (str, grammarKind))
                    text = inputArea.getText ();
```

251

```java
package caps.GraphEditor;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.Border;
import caps.Psdl.DataTypes;
import caps.Psdl.Vertex;
import caps.Parser.GrammarCheck;
import java.util.Vector;
import caps.Builder.*;
import java.io.StringReader;

/**
 * The main toolbar for the prototyping events.
 * Also handles the events associated with the toolbar buttons.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class ToolBar extends JToolBar implements ActionListener{

    /**
     * Initiates the 'Operator' event
     */
    private JButton operator = new JButton (new ImageIcon
("caps/Images/operator.gif"));

    /**
     * Initiates the 'Terminator' event
     */
    private JButton terminator = new JButton (new ImageIcon
("caps/Images/terminator.gif"));

    /**
     * Initiates the 'Stream' event
     */
    private JButton stream = new JButton (new ImageIcon
("caps/Images/streams.gif"));

    /**
     * Initiates the 'Select' event
     */
    private JButton select = new JButton (new ImageIcon
("caps/Images/select.gif"));

    /**
     * Initiates the 'Types' event
     */
    private JButton types = new JButton (new ImageIcon
("caps/Images/types.gif"));

    /**
     * Initiates the 'Parent Specs' event
     */
    private JButton parentSpecs = new JButton (new ImageIcon
("caps/Images/parentSpec.gif"));

    /**
     * Initiates the 'Timers' event
     */
    private JButton timers = new JButton (new ImageIcon
("caps/Images/timers.gif"));

    /**
     * Initiates the 'Graph Desc' event
     */
    private JButton graphDesc = new JButton (new ImageIcon
("caps/Images/graphDesc.gif"));

    /**
     * the JFrame that is the owner of this toolbar.
     */
    protected Editor parentFrame;

    /**
     * Constructs a new ToolBar object
     *
     * @param frame The parent frame of this toolbar object.
     */
    public ToolBar (Editor frame)
    {
        parentFrame = frame;

        add (operator);
        add (terminator);
        add (stream);
        add (select);
        add (types);
        add (parentSpecs);
        add (timers);
        add (graphDesc);

        operator.setToolTipText ("Draw an operator");
        terminator.setToolTipText ("Draw a terminator");
        stream.setToolTipText ("Draw a stream");
        select.setToolTipText ("Select");
        types.setToolTipText ("Types");
        parentSpecs.setToolTipText ("Parent specifications");
        timers.setToolTipText ("timers");
        graphDesc.setToolTipText ("Graph Description");

        operator.setFocusPainted (false);
```

```java
        operator.setActionCommand ("Draws an operator into the drawing
area");
        terminator.setActionCommand ("Draws a terminator into the drawing
area");
        stream.setActionCommand ("Draws a stream into the drawing area");
        select.setActionCommand ("Selects a component from the drawing
area");
        types.setActionCommand ("Opens the text editor to edit data types");
        parentSpecs.setActionCommand ("Opens the text editor to edit parent
specifications");
        timers.setActionCommand ("Opens the id list editor to edit timers");
        graphDesc.setActionCommand ("Opens the text editor to edit the graph
description");

        operator.addMouseListener (parentFrame.getStatusBar ());
        terminator.addMouseListener (parentFrame.getStatusBar ());
        stream.addMouseListener (parentFrame.getStatusBar ());
        select.addMouseListener (parentFrame.getStatusBar ());
        types.addMouseListener (parentFrame.getStatusBar ());
        parentSpecs.addMouseListener (parentFrame.getStatusBar ());
        timers.addMouseListener (parentFrame.getStatusBar ());
        graphDesc.addMouseListener (parentFrame.getStatusBar ());

        operator.addActionListener (this);
        terminator.addActionListener (this);
        stream.addActionListener (this);
        select.addActionListener (this);
        types.addActionListener (this);
        parentSpecs.addActionListener (this);
        timers.addActionListener (this);
        graphDesc.addActionListener (this);
    }

    /**
     * This method is called after another operation is finished
associated
     * with another button in the toolbar.
     * For example, When an operator is drawn on the DrawPanel, the
toolbar will go into
     * select mode.
     */
    public void enableSelectButton ()
    {
        select.requestFocus ();
    }

    public void setOperatorButton (boolean flag)
    {
        if (flag)
            operator.setEnabled (true);
        else
            operator.setEnabled (false);
    }

    /**
     * Handles the action events that occur when one of the buttons
     * in this toolbar is selected
     *
     * @param e The associated ActionEvent
     */
    public void actionPerformed (ActionEvent e)
    {
        if (e.getSource () == operator) {
            System.out.println ("Operator");             // *** Take out ***
            operator.setFocusPainted (true);
            parentFrame.getDrawPanel ().setSelectMode (false);
            parentFrame.getDrawPanel ().setCurrentComponent
(DrawPanel.OPERATOR);
        }
        else if (e.getSource () == terminator) {
            System.out.println ("Terminator");           // *** Take out ***
            parentFrame.getDrawPanel ().setSelectMode (false);
            parentFrame.getDrawPanel ().setCurrentComponent
(DrawPanel.TERMINATOR);
        }
        else if (e.getSource () == stream) {
            System.out.println ("Stream");
            parentFrame.getDrawPanel ().setSelectMode (false);
            parentFrame.getDrawPanel ().setCurrentComponent
(DrawPanel.STREAM);
        }
        else if (e.getSource () == select) {
            parentFrame.getDrawPanel ().setSelectMode (true);
        }
        else if (e.getSource () == types) {
            DataTypes types = parentFrame.getDataTypes ();
            TextEditor.openDialog ("Data Types", "View or Edit Data Types",
                                    types.toString (),
            GrammarCheck.DATA_TYPE, true);
            types.buildTypes (TextEditor.getString ());
        }
        else if (e.getSource () == parentSpecs) {
            Vertex parent = parentFrame.getDrawPanel ().getParentVertex ();
            TextEditor.openDialog ("Parent Vertex Specification", "View or
Edit Parent Specification",
                                    parent.getSpecification (false),
            GrammarCheck.CHECK_PARENT_SPEC, false);
            PsdlBuilder.setCurrentOp (parent);
            PsdlBuilder.ReInit (new StringReader (TextEditor.getString ()));
            try {
                PsdlBuilder.operator_spec ();
            } catch (ParseException ex) { /* This is already caught in
GrammarCheck */ }
        }
```

253

```java
        else if (e.getSource () == timers) {
            Vertex parent = parentFrame.getDrawPanel ().getParentVertex ();
            IdListEditor.openDialog (parent.getTimerList ());
            parent.setTimerList (IdListEditor.getIDList ());
        }
        else if (e.getSource () == graphDesc) {
            Vertex parent = parentFrame.getDrawPanel ().getParentVertex ();
            TextEditor.openDialog ("Informal Graph Description", "View or
Edit Informal Graph Description",
                        parent.getGraphDesc (),
GrammarCheck.INFORMAL_DESCRIPTION, true);
            parent.setGraphDesc (TextEditor.getString ());
        }

    }

} // End of the class ToolBar.
```

```java
package caps.GraphEditor;

import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.tree.*;
import caps.Psdl.*;

/**
 * The treepanel is the place where the hierarchic structure of
 * the prototype is displayed.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class TreePanel extends JTree implements TreeSelectionListener,
TreeModelListener {

    /**
     * the JFrame that is the owner of this panel.
     */
    Editor parentFrame;

    DefaultTreeModel model;

    /**
     * Constructs a new TreePanel object
     *
     * @param frame The parent frame of this treepanel object.
     */
    public TreePanel (Editor frame, Vertex root)
    {
        super ();

        parentFrame = frame;

        DefaultTreeCellRenderer renderer = new DefaultTreeCellRenderer ();
        setCellRenderer (renderer);
        addTreeSelectionListener (this);

        model = new DefaultTreeModel (root);
        model.setAsksAllowsChildren (true);    // Should show only the
composite ones
        model.addTreeModelListener (this);

        setModel (model);
        setCellRenderer (new TreePanelRenderer ());

        getSelectionModel ().setSelectionMode
(TreeSelectionModel.SINGLE_TREE_SELECTION);
        setShowsRootHandles (true);
        setEditable (false);
```

254

```java
        setAlignmentX (LEFT_ALIGNMENT);
        setAlignmentY (TOP_ALIGNMENT);
        setBorder (BorderFactory.createEtchedBorder ());
    }

    public void addNewDFC (DataFlowComponent dfc, DataFlowComponent parent)
    {
        System.out.println (parent.getChildCount ());
        int [] index = {parent.getIndex (dfc)};
        model.nodesWereInserted (parent, index);
    }

    public void removeDfc (DataFlowComponent dfc)
    {
        model.reload ();
    }

    public void valueChanged (TreeSelectionEvent e)
    {
        System.out.println ("Inside value changed");
        TreePath path = e.getPath ();
        DataFlowComponent dfc = (DataFlowComponent)
path.getLastPathComponent ();
        if (dfc.isRoot ())
            parentFrame.getDrawPanel ().gotoRoot ();
        else if (dfc instanceof Vertex && !(dfc instanceof External)) {
            if (((Vertex) dfc).isLeaf ()) {  // If this is an atomic vertex
                parentFrame.getDrawPanel ().changeLevel ((Vertex)
dfc.getParent ());
                parentFrame.getDrawPanel ().setSelectedDFC (dfc);
                parentFrame.getDrawPanel ().decompose ();
            }
            else {  // If this is composite
                parentFrame.getDrawPanel ().changeLevel ((Vertex) dfc);
            }
        }
        else if (dfc instanceof Edge) {
// If this is an Edge
            parentFrame.getDrawPanel ().changeLevel ((Vertex) dfc.getParent
());
            parentFrame.getDrawPanel ().setSelectedDFC (dfc);
        }
        parentFrame.getDrawPanel ().setMenuBarItems ();
    }

    public void treeNodesChanged (TreeModelEvent e)
    {
        System.out.println ("Inside tree nodes changed");
    }

    public void treeNodesInserted (TreeModelEvent e)
    {
            System.out.println ("Inside tree nodes inserted");
        }

    public void treeNodesRemoved (TreeModelEvent e)
    {
            System.out.println ("Inside tree nodes removed");
        }

    public void treeStructureChanged (TreeModelEvent e)
    {
            System.out.println ("Inside tree structure changed");
        }

    } // End of the class TreePanel.
```

255

```java
package caps.GraphEditor;

import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JTree;
import javax.swing.tree.TreeCellRenderer;
import javax.swing.tree.DefaultMutableTreeNode;
import java.awt.Component;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import caps.Psdl.*;

public class TreePanelRenderer extends JLabel implements TreeCellRenderer
{

    static protected Font defaultFont;

    static protected ImageIcon termCompositeIcon;

    static protected ImageIcon termAtomicIcon;

    static protected ImageIcon opCompositeIcon;

    static protected ImageIcon opAtomicIcon;

    static protected ImageIcon streamIcon;

    static protected ImageIcon stateStreamIcon;

    /** Color to use for the background when selected. */
    static protected final Color SelectedBackgroundColor =
Color.lightGray;//new Color(0, 0, 128);

    static
    {
    try {
        defaultFont = new Font("SansSerif", 0, 12);
    } catch (Exception e) {}
    try {
        termCompositeIcon = new ImageIcon
("caps/Images/termComposite.gif");
        termAtomicIcon = new ImageIcon ("caps/Images/termAtomic.gif");
        opCompositeIcon = new ImageIcon ("caps/Images/opComposite.gif");
        opAtomicIcon = new ImageIcon ("caps/Images/opAtomic.gif");
        streamIcon = new ImageIcon ("caps/Images/streamIcon.gif");
        stateStreamIcon = new ImageIcon
("caps/Images/stateStreamIcon.gif");
    } catch (Exception e) {
        System.out.println("Couldn't load images: " + e);
    }
    }

    /** Whether or not the item that was last configured is selected. */
    protected boolean selected;

    /**
     * This is messaged from JTree whenever it needs to get the size
     * of the component or it wants to draw it.
     * This attempts to set the font based on value, which will be
     * a TreeNode.
     */
    public Component getTreeCellRendererComponent(JTree tree, Object
value,
                            boolean selected, boolean expanded,
                            boolean leaf, int row,
                            boolean hasFocus) {

    Font font;
    String stringValue = tree.convertValueToText(value, selected,
                            expanded, leaf, row, hasFocus);

    /* Set the color and the font based on the SampleData userObject. */
    DataFlowComponent userObject = (DataFlowComponent) value;

    /* Set the text. */
    setText(stringValue);
    /* Tooltips used by the tree. */
    setToolTipText(stringValue);

    /* Set the image. */
    if (userObject instanceof Vertex && ((Vertex) userObject).isLeaf ())
    {

        if (((Vertex) userObject).isTerminator ())
            setIcon (termAtomicIcon);
        else
            setIcon (opAtomicIcon);

    }
    else if (userObject instanceof Vertex && !((Vertex)
userObject).isLeaf ()) {
        if (((Vertex) userObject).isTerminator ())
            setIcon (termCompositeIcon);
        else
            setIcon (opCompositeIcon);

    }
    else if (userObject instanceof Edge && ((Edge)
userObject).isStateStream ())
        setIcon (stateStreamIcon);
    else
        setIcon (streamIcon);

    /*

    if(hasFocus)
        setForeground(Color.cyan);
    else
```

256

```java
package caps.GraphEditor;

import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import caps.Psdl.*;
import caps.Display.DisplayVertex;
import caps.Parser.GrammarCheck;
import java.util.Vector;
import java.util.Enumeration;

public class VertexProperties extends JDialog implements ActionListener {

    public static final int TO_OPERATOR = 0;

    public static final int TO_TERMINATOR = 1;

    public static final int UNCHANGED = 3;

    private int changeStatus;

    Vertex targetVertex;

    DisplayVertex dVertex;

    JPanel namePanel;
    JPanel triggerPanel;
    JPanel timingPanel;
    JPanel guardsPanel;
    JPanel keywordsPanel;
    JPanel okPanel;

    JTextField nameField;
    TextArea ifCondField;
    JTextField metField;
    JTextField periodField;
    JTextField fwField;

    JLabel metLabel;
    JLabel periodLabel;
    JLabel finishWithinLabel;

    JComboBox operatorCombo;
    JComboBox languageCombo;
    JComboBox triggerCombo;
    JComboBox timingCombo;
    JComboBox metUnitsCombo;
    JComboBox periodUnitsCombo;
    JComboBox fwUnitsCombo;

    JButton ifConditionButton;
    JButton triggerReqByButton;
```

```java
        setForeground(userObject.getColor());
    if(userObject.getFont() == null)
        setFont(defaultFont);
    else
        setFont(userObject.getFont());

    /* Update the selected flag for the next paint. */
    this.selected = selected;

    return this;
}

/**
 * paint is subclassed to draw the background correctly.  JLabel
 * currently does not allow backgrounds other than white, and it
 * will also fill behind the icon.  Something that isn't desirable.
 */
public void paint(Graphics g) {
    Color    bColor;
    Icon     currentI = getIcon();

    if(selected)
        bColor = SelectedBackgroundColor;
    else if(getParent() != null)
        /* Pick background color up from parent (which will come from
           the JTree we're contained in). */
        bColor = getParent().getBackground();
    else
        bColor = getBackground();
    g.setColor(bColor);
    if(currentI != null && getText() != null) {
        int     offset = (currentI.getIconWidth() +
getIconTextGap());
        g.fillRect(offset, 0, getWidth() - 1 - offset,
            getHeight() - 1);
    }
    else
        g.fillRect(0, 0, getWidth()-1, getHeight()-1);
    super.paint(g);
}
```

257

```java
JButton metReqByButton;
JButton periodReqByButton;
JButton fwReqByButton;
JButton outputGuardsButton;
JButton exceptionGuardsButton;
JButton exceptionListButton;
JButton timerOpsButton;
JButton keywordsButton;
JButton informalDescButton;
JButton formalDescButton;
JButton okButton;
JButton cancelButton;
JButton helpButton;
JButton triggerStreamsButton;

Editor parentFrame;

Vertex tempVertex;

public VertexProperties (Editor parent)
{
    super (parent, "Vertex Properties", true);
    parentFrame = parent;
    setResizable (false);
    initialize ();
    pack ();
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    setLocation ((screenSize.width - getWidth ()) / 2,
                 (screenSize.height - getHeight ()) / 2);
}

public void initialize ()
{
    Box box = Box.createVerticalBox ();

    GridBagConstraints gbc = new GridBagConstraints ();
    gbc.fill = GridBagConstraints.BOTH;
    gbc.insets = new Insets (1, 2, 1, 2);

    namePanel = new JPanel (new GridBagLayout ());
    namePanel.setBorder (BorderFactory.createTitledBorder (""));
    nameField = new JTextField ();
    operatorCombo = new JComboBox ();
    operatorCombo.addItem ("Operator");
    operatorCombo.addItem ("Terminator");
    operatorCombo.addActionListener (this);
    languageCombo = new JComboBox ();
    languageCombo.addItem ("Ada");
    languageCombo.addItem ("TAE");
    gbc.gridwidth = 1; gbc.gridheight = 1; gbc.gridx = 0; gbc.gridy = 0;
    namePanel.add (new JLabel ("Name :"), gbc);
    gbc.gridwidth = 2; gbc.gridx++;

    namePanel.add (nameField, gbc);
    gbc.gridwidth = 1; gbc.gridx = 3;
    namePanel.add (operatorCombo, gbc);
    gbc.gridwidth = 2; gbc.gridx = 0; gbc.gridy++;
    namePanel.add (new JLabel ("Implementation Language :"), gbc);
    gbc.gridwidth = 1; gbc.gridx = 2;
    namePanel.add (languageCombo, gbc);

    triggerPanel = new JPanel (new GridBagLayout ());
    triggerPanel.setBorder (BorderFactory.createTitledBorder ("Trigger :
"));

    triggerCombo = new JComboBox ();
    triggerCombo.addItem ("Unprotected");
    triggerCombo.addItem ("By Some");
    triggerCombo.addItem ("By All");
    triggerCombo.addActionListener (this);
    triggerStreamsButton = new JButton (" Stream List ");
    triggerStreamsButton.addActionListener (this);
    ifConditionButton = new JButton ("If Condition");
    ifConditionButton.addActionListener (this);
    ifCondField = new TextArea ("", 1, 20,
TextArea.SCROLLBARS_VERTICAL_ONLY);
    ifCondField.setEditable (false);
    triggerReqByButton = new JButton (" Required By ");
    triggerReqByButton.addActionListener (this);
    //gbc.gridwidth = 1; gbc.gridx = 0; gbc.gridy = 0;
    //triggerPanel.add (new JLabel ("Trigger :"), gbc);
    gbc.gridwidth = 1; gbc.gridx = 1; gbc.gridy = 0;
    triggerPanel.add (triggerCombo, gbc);
    gbc.gridx = 3;
    triggerPanel.add (triggerStreamsButton, gbc);
    gbc.gridx = 1; gbc.gridy++;
    triggerPanel.add (ifConditionButton, gbc);
    gbc.gridwidth = 2; gbc.gridx++;
    triggerPanel.add (ifCondField, gbc);
    //gbc.gridwidth = 1; gbc.gridx = 4;
    //triggerPanel.add (Box.createRigidArea (new Dimension (10, 5)));
    gbc.gridwidth = 1; gbc.gridx = 1; gbc.gridy = 2;
    triggerPanel.add (triggerReqByButton, gbc);

    timingPanel = new JPanel (new GridBagLayout ());
    timingPanel.setBorder (BorderFactory.createTitledBorder ("Timing :
"));

    timingCombo = new JComboBox ();
    timingCombo.addItem ("Non-time critical");
    timingCombo.addItem ("Periodic");
    timingCombo.addItem ("Sporadic");
    timingCombo.addActionListener (this);
    metField = new JTextField ();
    metUnitsCombo = getUnitsCombo ();
    metReqByButton = new JButton (" Required By ");
    metReqByButton.addActionListener (this);
```

```java
periodField = new JTextField ();
periodUnitsCombo = getUnitsCombo ();
periodReqByButton = new JButton (" Required By ");
periodReqByButton.addActionListener (this);
fwField = new JTextField ();
fwUnitsCombo = getUnitsCombo ();
fwReqByButton = new JButton (" Required By ");
fwReqByButton.addActionListener (this);
metLabel = new JLabel ("MET :");
periodLabel = new JLabel ("Period : ");
finishWithinLabel = new JLabel ("FinishWithin : ");
gbc.gridwidth = 1; gbc.gridx = 1; gbc.gridy = 0;
timingPanel.add (timingCombo, gbc);
gbc.gridwidth = 1; gbc.gridx = 0; gbc.gridy = 1;
timingPanel.add (metLabel, gbc);
gbc.gridwidth = 1; gbc.gridx = 1;
timingPanel.add (metField, gbc);
gbc.gridwidth = 1; gbc.gridx = 2;
timingPanel.add (metUnitsCombo, gbc);
gbc.gridwidth = 1; gbc.gridx = 3;
timingPanel.add (metReqByButton, gbc);
gbc.gridwidth = 1; gbc.gridx = 0; gbc.gridy = 2;
timingPanel.add (periodLabel, gbc);
gbc.gridwidth = 1; gbc.gridx = 1;
timingPanel.add (periodField, gbc);
gbc.gridwidth = 1; gbc.gridx = 2;
timingPanel.add (periodUnitsCombo, gbc);
gbc.gridwidth = 1; gbc.gridx = 3;
timingPanel.add (periodReqByButton, gbc);
gbc.gridwidth = 1; gbc.gridx = 0; gbc.gridy = 3;
timingPanel.add (finishWithinLabel, gbc);
gbc.gridwidth = 1; gbc.gridx = 1;
timingPanel.add (fwField, gbc);
gbc.gridwidth = 1; gbc.gridx = 2;
timingPanel.add (fwUnitsCombo, gbc);
gbc.gridwidth = 1; gbc.gridx = 3;
timingPanel.add (fwReqByButton, gbc);

gbc.insets = new Insets (1, 15, 1, 15);
guardsPanel = new JPanel (new GridBagLayout ());
guardsPanel.setBorder (BorderFactory.createTitledBorder (""));
outputGuardsButton = new JButton (" Output Guards ");
outputGuardsButton.addActionListener (this);
exceptionGuardsButton = new JButton (" Exception Guards ");
exceptionGuardsButton.addActionListener (this);
exceptionListButton = new JButton (" Exception List ");
exceptionListButton.addActionListener (this);
timerOpsButton = new JButton (" Timer Ops ");
timerOpsButton.addActionListener (this);
gbc.gridwidth = 2; gbc.gridx = 0; gbc.gridy = 0;
guardsPanel.add (outputGuardsButton, gbc);
gbc.gridx = 2;

guardsPanel.add (exceptionGuardsButton, gbc);
gbc.gridx = 0; gbc.gridy = 1;
guardsPanel.add (exceptionListButton, gbc);
gbc.gridx = 2;
guardsPanel.add (timerOpsButton, gbc);

gbc.insets = new Insets (1, 2, 1, 2);
keywordsPanel = new JPanel (new FlowLayout ());
keywordsPanel.setBorder (BorderFactory.createTitledBorder (""));
keywordsButton = new JButton (" Keywords ");
keywordsButton.addActionListener (this);
informalDescButton = new JButton (" Informal Desc ");
informalDescButton.addActionListener (this);
formalDescButton = new JButton (" Formal Desc ");
formalDescButton.addActionListener (this);
keywordsPanel.add (keywordsButton);
keywordsPanel.add (informalDescButton);
keywordsPanel.add (formalDescButton);

okPanel = new JPanel (new GridBagLayout ());
okButton = new JButton ("OK");
okButton.addActionListener (this);
cancelButton = new JButton ("Cancel");
cancelButton.addActionListener (this);
helpButton = new JButton ("Help");
helpButton.addActionListener (this);
gbc.gridwidth = 1; gbc.gridx = 1; gbc.gridy = 0;
okPanel.add (okButton, gbc);
gbc.gridwidth = 1; gbc.gridx = 4;
okPanel.add (cancelButton, gbc);
gbc.gridwidth = 1; gbc.gridx = 7;
okPanel.add (helpButton, gbc);

box.add (namePanel);
box.add (Box.createVerticalStrut (5));
box.add (triggerPanel);
box.add (Box.createVerticalStrut (5));
box.add (timingPanel);
box.add (Box.createVerticalStrut (2));
box.add (guardsPanel);
box.add (Box.createVerticalStrut (2));
box.add (keywordsPanel);
box.add (Box.createVerticalStrut (5));
box.add (okPanel);
box.add (Box.createVerticalStrut (3));

getContentPane ().add (box, BorderLayout.CENTER);
}

public JComboBox getUnitsCombo ()
{
    JComboBox c = new JComboBox ();
```

259

```java
        c.addItem ("microsec");
        c.addItem ("ms");
        c.addItem ("sec");
        c.addItem ("min");
        c.addItem ("hours");
        return c;
    }

    public void setVertex (Vertex v)
    {
        changeStatus = UNCHANGED;

        targetVertex = v;
        tempVertex = (Vertex) v.clone ();

        nameField.setText (v.getLabel ());
        operatorCombo.removeActionListener (this);
        operatorCombo.setSelectedItem ("Operator");
        if (v.isTerminator ())
            operatorCombo.setSelectedItem ("Terminator");
        operatorCombo.addActionListener (this);
        languageCombo.setSelectedItem (tempVertex.getImpLanguage ());
        triggerCombo.setSelectedIndex (tempVertex.getTriggerType ());
        timingCombo.removeActionListener (this);                    //
Otherwise it goes into actionperformed
        timingCombo.setSelectedIndex (targetVertex.getTimingType ()); // and
deletes the element of the vector
        timingCombo.addActionListener (this);

        resetTimingPanelComponents ();

        ifCondField.setText (v.getIfCondition ());

        if (triggerCombo.getSelectedIndex () == Vertex.UNPROTECTED)
            triggerStreamsButton.setEnabled (false);
        else
            triggerStreamsButton.setEnabled (true);

        PSDLTime met = tempVertex.getMet ();
        if (met != null) {
            metField.setText (String.valueOf (met.getTimeValue ()));
            metUnitsCombo.setSelectedIndex (met.getTimeUnits ());
        }
        else {
            metField.setText ("");
            metUnitsCombo.setSelectedIndex (1);
        }

        if (tempVertex.getTimingType () == Vertex.PERIODIC) {
            PSDLTime period = tempVertex.getPeriod ();
            PSDLTime fw = tempVertex.getFinishWithin ();
            periodLabel.setText ("Period : ");
            finishWithinLabel.setText ("Finish Within : ");

            if (period != null) {
                periodField.setText (String.valueOf (period.getTimeValue ()));
                periodUnitsCombo.setSelectedIndex (period.getTimeUnits ());
            }
            else {
                periodField.setText ("");
                periodUnitsCombo.setSelectedIndex (1);
            }

            if (fw != null) {
                fwField.setText (String.valueOf (fw.getTimeValue ()));
                fwUnitsCombo.setSelectedIndex (fw.getTimeUnits ());
            }
            else {
                fwField.setText ("");
                fwUnitsCombo.setSelectedIndex (1);
            }
        }
        else if (tempVertex.getTimingType () == Vertex.SPORADIC) {
            PSDLTime mcp = tempVertex.getMcp ();
            PSDLTime mrt = tempVertex.getMrt ();
            periodLabel.setText ("MCP : ");
            finishWithinLabel.setText ("MRT : ");
            if (mcp != null) {
                periodField.setText (String.valueOf (mcp.getTimeValue ()));
                periodUnitsCombo.setSelectedIndex (mcp.getTimeUnits ());
            }
            else {
                periodField.setText ("");
                periodUnitsCombo.setSelectedIndex (1);
            }

            if (mrt != null) {
                fwField.setText (String.valueOf (mrt.getTimeValue ()));
                fwUnitsCombo.setSelectedIndex (mrt.getTimeUnits ());
            }
            else {
                fwField.setText ("");
                fwUnitsCombo.setSelectedIndex (1);
            }
        }
        else {
            fwUnitsCombo.setSelectedIndex (1);
            periodUnitsCombo.setSelectedIndex (1);
        }

        if (tempVertex.isTerminator ()) {
            metLabel.setEnabled (false);
            metField.setEnabled (false);
            metUnitsCombo.setEnabled (false);
        }
    }

    public void setDisplayVertex (DisplayVertex v)
    {
```

260

```
        setButtonText (keywordsButton, tempVertex.getKeywordList ());
        setButtonText (informalDescButton, tempVertex.getInformalDesc ());
        setButtonText (formalDescButton, tempVertex.getFormalDesc ());
    }

    public void actionPerformed (ActionEvent e)
    {
        if (e.getSource () == ifConditionButton) {
            TextEditor.openDialog ("Operator Trigger If Condition", "View or
Edit Operator Trigger If Condition",
                                    ifCondField.getText (),
GrammarCheck.EXPRESSION, true);
            ifCondField.setText (TextEditor.getString ());
            tempVertex.setIfCondition (ifCondField.getText ());
        }
        else if (e.getSource () == triggerReqByButton) {
            IdListEditor.openDialog (tempVertex.getTriggerReqmts ());
            tempVertex.setTriggerReqmts (IdListEditor.getIDList ());
            setButtonText (triggerReqByButton, IdListEditor.getIDList ());
        }
        else if (e.getSource () == triggerStreamsButton) {
            IdListEditor.openDialog (tempVertex.getTriggerStreamsList ());
            tempVertex.setTriggerStreamsList (IdListEditor.getIDList ());
            setButtonText (triggerStreamsButton, IdListEditor.getIDList ());
        }
        else if (e.getSource () == metReqByButton) {
            IdListEditor.openDialog (tempVertex.getMetReqmts ());
            tempVertex.setMetReqmts (IdListEditor.getIDList ());
            setButtonText (metReqByButton, IdListEditor.getIDList ());
        }
        else if (e.getSource () == periodReqByButton) {
            if (timingCombo.getSelectedIndex () == Vertex.PERIODIC) {
                IdListEditor.openDialog (tempVertex.getPeriodReqmts ());
                tempVertex.setPeriodReqmts (IdListEditor.getIDList ());
            }
            else if (timingCombo.getSelectedIndex () == Vertex.SPORADIC) {
                IdListEditor.openDialog (tempVertex.getMcpReqmts ());
                tempVertex.setMcpReqmts (IdListEditor.getIDList ());
            }
            setButtonText (periodReqByButton, IdListEditor.getIDList ());
        }
        else if (e.getSource () == fwReqByButton) {
            if (timingCombo.getSelectedIndex () == Vertex.PERIODIC) {
                IdListEditor.openDialog (tempVertex.getFinishWithinReqmts ());
                tempVertex.setFinishWithinReqmts (IdListEditor.getIDList ());
            }
            else if (timingCombo.getSelectedIndex () == Vertex.SPORADIC) {
                IdListEditor.openDialog (tempVertex.getMrtReqmts ());
                tempVertex.setMrtReqmts (IdListEditor.getIDList ());
            }
            setButtonText (fwReqByButton, IdListEditor.getIDList ());
        }

        dVertex = v;
        setVisible (true);
    }

    public void resetTimingPanelComponents ()
    {
        metLabel.setEnabled (true);
        if (!tempVertex.isTerminator ())
            metField.setText ("");
        metField.setEnabled (true);
        metUnitsCombo.setEnabled (true);
        metReqByButton.setEnabled (true);
        periodLabel.setEnabled (true);
        periodField.setText ("");
        periodField.setEnabled (true);
        periodUnitsCombo.setEnabled (true);
        periodReqByButton.setEnabled (true);
        finishWithinLabel.setEnabled (true);
        fwField.setText ("");
        fwField.setEnabled (true);
        fwUnitsCombo.setEnabled (true);
        fwReqByButton.setEnabled (true);
        if (timingCombo.getSelectedIndex () == Vertex.NON_TIME_CRITICAL) {
            periodLabel.setEnabled (false);
            periodField.setEnabled (false);
            periodUnitsCombo.setEnabled (false);
            periodReqByButton.setEnabled (false);
            finishWithinLabel.setEnabled (false);
            fwField.setEnabled (false);
            fwUnitsCombo.setEnabled (false);
            fwReqByButton.setEnabled (false);
            if (!tempVertex.isTerminator ()) {
                metLabel.setEnabled (false);
                metField.setEnabled (false);
                metUnitsCombo.setEnabled (false);
                metReqByButton.setEnabled (false);
            }
        }
        setButtonText (triggerReqByButton, tempVertex.getTriggerReqmts ());
        setButtonText (triggerStreamsButton,
tempVertex.getTriggerStreamsList ());
        setButtonText (metReqByButton, tempVertex.getMetReqmts ());
        setButtonText (periodReqByButton, tempVertex.getPeriodReqmts ());
        setButtonText (periodReqByButton, tempVertex.getMcpReqmts ());
        setButtonText (fwReqByButton, tempVertex.getFinishWithinReqmts ());
        setButtonText (fwReqByButton, tempVertex.getMrtReqmts ());
        setButtonText (outputGuardsButton, tempVertex.getOutputGuardList
());
        setButtonText (exceptionGuardsButton,
tempVertex.getExceptionGuardList ());
        setButtonText (exceptionListButton, tempVertex.getExceptionList ());
        setButtonText (timerOpsButton, tempVertex.getTimerOpList ());
    }
```

```java
      else if (e.getSource () == outputGuardsButton) {
         TextEditor.openDialog ("Operator Output Guard", "View or Edit
Operator Output Guard Equation",
                    tempVertex.getOutputGuardList (),
GrammarCheck.CHECK_OUTPUT_GUARDS, true);
         tempVertex.setOutputGuardList (TextEditor.getString ());
         setButtonText (outputGuardsButton, TextEditor.getString ());
      }
      else if (e.getSource () == exceptionGuardsButton) {
         TextEditor.openDialog ("Operator Exceptions", "View or Edit
Operator Exceptions",
                    tempVertex.getExceptionGuardList (),
GrammarCheck.CHECK_EXCEPTION_GUARDS, true);
         tempVertex.setExceptionGuardList (TextEditor.getString ());
         setButtonText (exceptionGuardsButton, TextEditor.getString ());
      }
      else if (e.getSource () == exceptionListButton) {
         TextEditor.openDialog ("Operator Exceptions", "View or Edit
Operator Exceptions",
                    tempVertex.getExceptionList (),
GrammarCheck.CHECK_EXCEPTION_LIST, true);
         tempVertex.setExceptionList (TextEditor.getString ());
         setButtonText (exceptionListButton, TextEditor.getString ());
      }
      else if (e.getSource () == timerOpsButton) {
         TextEditor.openDialog ("Operator Timers", "View or Edit Operator
Timers",
                    tempVertex.getTimerOpList (),
GrammarCheck.CHECK_TIMER_OPS, true);
         tempVertex.setTimerOpList (TextEditor.getString ());
         setButtonText (timerOpsButton, TextEditor.getString ());
      }
      else if (e.getSource () == keywordsButton) {
         IdListEditor.openDialog (tempVertex.getKeywordList ());
         tempVertex.setKeywordList (IdListEditor.getIDList ());
         setButtonText (keywordsButton, IdListEditor.getIDList ());
      }
      else if (e.getSource () == informalDescButton) {
         TextEditor.openDialog ("Informal Design Description", "View or
Edit Informal Description",
                    tempVertex.getInformalDesc (),
GrammarCheck.INFORMAL_DESCRIPTION, true);
         tempVertex.setInformalDesc (TextEditor.getString ());
         setButtonText (informalDescButton, TextEditor.getString ());
      }
      else if (e.getSource () == formalDescButton) {
         TextEditor.openDialog ("Formal Design Description", "View or Edit
Formal Description",
                    tempVertex.getFormalDesc (),
GrammarCheck.FORMAL_DESCRIPTION, true);
         tempVertex.setFormalDesc (TextEditor.getString ());
         setButtonText (formalDescButton, TextEditor.getString ());
      }
      else if (e.getSource () == okButton) {
         boolean exceptionOccurred = false;
         String str = nameField.getText ();
         if (!GrammarCheck.isValid (str, GrammarCheck.ID)) {
            showErrorDialog ("Illegal vertex name");
            exceptionOccurred = true;
         }
         if (timingCombo.getSelectedIndex () != Vertex.NON_TIME_CRITICAL)
         {
            str = metField.getText ();
            if (!(targetVertex.isTerminator ())) {
               if (!GrammarCheck.isValid (str,
GrammarCheck.INTEGER_LITERAL) {
                  showErrorDialog ("Illegal value for met field");
                  exceptionOccurred = true;
            }
            str = periodField.getText ();
            if (str.length () != 0) {
               if (!GrammarCheck.isValid (str,
GrammarCheck.INTEGER_LITERAL) {
                  if (timingCombo.getSelectedIndex () == Vertex.PERIODIC)
                     showErrorDialog ("Illegal value for period field");
                  else
                     showErrorDialog ("Illegal value for mcp field");
                  exceptionOccurred = true;
            }
            str = fwField.getText ();
            if (str.length () != 0) {
               if (!GrammarCheck.isValid (str,
GrammarCheck.INTEGER_LITERAL) {
                  if (timingCombo.getSelectedIndex () == Vertex.PERIODIC)
                     showErrorDialog ("Illegal value for finish within
field");
                  else
                     showErrorDialog ("Illegal value for mrt field");
                  exceptionOccurred = true;
            }
         }
         if (!exceptionOccurred) {
            targetVertex.setLabel (nameField.getText ());
            targetVertex.setImpLanguage ((String)
languageCombo.getSelectedItem ());
            targetVertex.setTriggerType (triggerCombo.getSelectedIndex
());
            if (triggerCombo.getSelectedIndex () != Vertex.UNPROTECTED)
```

```java
        else
            targetVertex.setTriggerStreamsList
(tempVertex.getTriggerStreamsList ());
        else
            targetVertex.setTriggerStreamsList (new Vector ());

        targetVertex.setTimingType (timingCombo.getSelectedIndex ());
        if (timingCombo.getSelectedIndex () !=
Vertex.NON_TIME_CRITICAL &&          !(targetVertex.isTerminator ())) {
            targetVertex.setMet (new PSDLTime (Integer.parseInt
(metField.getText ()),          metUnitsCombo.getSelectedIndex
()));
            targetVertex.setMetReqmts (tempVertex.getMetReqmts ());
        }
        else if (!(targetVertex.isTerminator ())) {
            targetVertex.setMet (null);
            targetVertex.getMetReqmts ().removeAllElements ();
        }

        if (timingCombo.getSelectedIndex () == Vertex.PERIODIC) {
            if (periodField.getText ().length () != 0) {
                targetVertex.setPeriod (new PSDLTime (Integer.parseInt
(periodField.getText ()),
periodUnitsCombo.getSelectedIndex ()));
                targetVertex.setPeriodReqmts (tempVertex.getPeriodReqmts
());
            }
            else {
                targetVertex.setPeriod (null);
                targetVertex.getPeriodReqmts ().removeAllElements ();
            }
            if (fwField.getText ().length () != 0) {
                targetVertex.setFinishWithin (new PSDLTime
(Integer.parseInt (fwField.getText ()),
fwUnitsCombo.getSelectedIndex ()));
                targetVertex.setFinishWithinReqmts
(tempVertex.getFinishWithinReqmts ());
            }
            else {
                targetVertex.setFinishWithin (null);
                targetVertex.getFinishWithinReqmts ().removeAllElements
();
            }
        }
        else if (timingCombo.getSelectedIndex () == Vertex.SPORADIC) {
            if (periodField.getText ().length () != 0) {
                targetVertex.setMcp (new PSDLTime (Integer.parseInt
(periodField.getText ()),
periodUnitsCombo.getSelectedIndex ()));
                targetVertex.setMcpReqmts (tempVertex.getMcpReqmts ());
            }
            else {
                targetVertex.setMcp (null);
                targetVertex.getMcpReqmts ().removeAllElements ();
            }
            if (fwField.getText ().length () != 0) {
                targetVertex.setMrt (new PSDLTime (Integer.parseInt
(fwField.getText ()),
                                fwUnitsCombo.getSelectedIndex
()));
                targetVertex.setMrtReqmts (tempVertex.getMrtReqmts ());
            }
            else {
                targetVertex.setMrt (null);
                targetVertex.getMrtReqmts ().removeAllElements ();
            }
            targetVertex.getPeriodReqmts ().removeAllElements ();
            targetVertex.getFinishWithinReqmts ().removeAllElements ();
        }

        targetVertex.setIfCondition (tempVertex.getIfCondition ());
        targetVertex.setOutputGuardList (tempVertex.getOutputGuardList
());

        targetVertex.setExceptionGuardList
(tempVertex.getExceptionGuardList ());
        targetVertex.setExceptionList (tempVertex.getExceptionList
());

        targetVertex.setTimerOpList (tempVertex.getTimerOpList ());
        targetVertex.setInformalDesc (tempVertex.getInformalDesc ());
        targetVertex.setFormalDesc (tempVertex.getFormalDesc ());
        targetVertex.setTriggerReqmts (tempVertex.getTriggerReqmts
());

        targetVertex.setKeywordList (tempVertex.getKeywordList ());

        dVertex.setLabelShape ((Graphics2D) parentFrame.getDrawPanel
().getGraphics ());
        dVertex.setMetShape ((Graphics2D) parentFrame.getDrawPanel
().getGraphics ());

        setVisible (false);

        if (changeStatus == TO_OPERATOR) {
            targetVertex.setTerminator (false);
            parentFrame.getDrawPanel ().changeLevel ((Vertex)
targetVertex.getParent ());
        }
        else if (changeStatus == TO_TERMINATOR) {
```

263

```java
            DataFlowComponent dfc;
            for (Enumeration enum =
targetVertex.breadthFirstEnumeration (); enum.hasMoreElements ();) {
            dfc = (DataFlowComponent) enum.nextElement ();
            if (dfc instanceof Vertex) {
                ((Vertex) dfc).setTerminator (true);
                ((Vertex) dfc).setMet (new PSDLTime (0,
PSDLTime.ms));
            }
        }
        parentFrame.getDrawPanel ().changeLevel ((Vertex)
targetVertex.getParent ());
        }
        else {
            parentFrame.getDrawPanel ().clearAllComponentsFromScreen
(null);  // Is there a better way\
        parentFrame.getDrawPanel ().paint (parentFrame.getDrawPanel
().getGraphics ()); // Is there a better way\
        parentFrame.getTreePanel ().repaint ();
        parentFrame.setSaveRequired (true);
        }
    }
    else if (e.getSource () == cancelButton) {
        setVisible (false);
    }
    else if (e.getSource () == helpButton) {
    }
    else if (e.getSource () == operatorCombo) {
        changeStatus = operatorCombo.getSelectedIndex ();
        if (changeStatus == TO_TERMINATOR) {
            metField.setText ("0");
            metLabel.setEnabled (false);
            metField.setEnabled (false);
            metUnitsCombo.setEnabled (false);
            metUnitsCombo.setSelectedIndex (PSDLTime.ms);
        }
        else if (changeStatus == TO_OPERATOR) {
            metField.setEnabled (true);
            metLabel.setEnabled (true);
            metUnitsCombo.setEnabled (true);
            if (targetVertex.getMet () != null) {
                metField.setText (String.valueOf (targetVertex.getMet
().getTimeValue ()));
                metUnitsCombo.setSelectedIndex (targetVertex.getMet
().getTimeUnits ());
            }
            else {           // It does not have met
                metField.setText ("");
                metUnitsCombo.setSelectedIndex (PSDLTime.ms);
            }
        }

        if (targetVertex.isTerminator () && changeStatus == TO_TERMINATOR)
||        !targetVertex.isTerminator () && changeStatus == TO_OPERATOR) {
        // Restore ro older one
            changeStatus = UNCHANGED;
    }
    else if (e.getSource () == triggerCombo) {
        if (triggerCombo.getSelectedIndex () == Vertex.UNPROTECTED)
            triggerStreamsButton.setEnabled (false);
        else
            triggerStreamsButton.setEnabled (true);
    }
    else if (e.getSource () == timingCombo) {
        System.out.println ("Inside timingcombo");
        resetTimingPanelComponents ();
        tempVertex.getMetReqmts ().removeAllElements ();
        tempVertex.getPeriodReqmts ().removeAllElements ();
        tempVertex.getFinishWithinReqmts ().removeAllElements ();
        tempVertex.getMrtReqmts ().removeAllElements ();
        tempVertex.getMcpReqmts ().removeAllElements ();

        if (timingCombo.getSelectedIndex () == Vertex.SPORADIC) {
            periodLabel.setText ("MCP :            ");
            finishWithinLabel.setText ("MRT :            ");
        }
        else {
            periodLabel.setText ("Period : ");
            finishWithinLabel.setText ("Finish within : ");
        }
        if (tempVertex.isTerminator ()) {
            metLabel.setEnabled (false);
            metField.setEnabled (false);
            metUnitsCombo.setEnabled (false);
        }
    }
}

public void showErrorDialog (String str)
{
    JOptionPane.showMessageDialog (this, str, "Error Message",
JOptionPane.ERROR_MESSAGE);
}

public void setButtonText (JButton b, Object o)
{
    if (((o instanceof Vector) && (((Vector) o).size () != 0)) ||
        ((o instanceof String) && (((String) o).length () != 0))) {
        if (!b.getText ().endsWith ("..."))
            b.setText (b.getText ().trim () + " ...");
    }
    else {
        if (b.getText ().endsWith ("..."))
```

264

```java
() .length () - 4) + " ");
      b.setText (" " + b.getText () .substring (0, b.getText
   }
}

} // End of the class VertexProperties


package caps.Psdl;

import javax.swing.tree.DefaultMutableTreeNode;
import java.awt.*;
import caps.GraphEditor.FontConstants;

/**
 * DataFlowComponent is the abstract base class of the Vertex and
 * Edge classes.
 * It extends DefaultMutableTreeNode, so every object of this class is
actually a tree node.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public abstract class DataFlowComponent extends DefaultMutableTreeNode {

/**
 * The label to display on the DrawPanel
 */
protected String label;

/**
 * The unique id of components.
 */
protected static int UNIQUE_ID = 0;

/**
 * The id of this component
 */
protected int id;                    // Op-_num or edge id

/**
 * The font parameter of the label.
 */
protected int labelFont;

/**
 * The font representation of the label.
 */
protected Font lFont;

/**
 * The x-offset of the label from the center of the component
 */
protected int labelXOffset;

/**
 * The y-offset of the label from the center of the component
 */
protected int labelYOffset;
```

265

```java
/**
 * The met of a Vertex or the latency of a Stream.
 */
protected PSDLTime met;

/**
 * The font parameter of the met label of this component.
 */
protected int metFont;

/**
 * The x-offset of the met label from the center of this component.
 */
protected int metXOffset;

/**
 * The y-offset of the met label from the center of this component.
 */
protected int metYOffset;

/**
 * The font representation of the met (or latency).
 */
protected Font metlFont;              // The real font to display on the
screen

/**
 * The parent of this component.
 */
//protected Vertex parent;

/**
 * The constructor for this class.
 *
 * @param v       The parent vertex of this component
 */
protected DataFlowComponent (Vertex v)
{
    super ();

    labelXOffset = 0;
    labelYOffset = 0;

    metXOffset = 0;
    metYOffset = -40;

    labelFont = 4;
    metFont = 4;
    lFont = new Font ("Courier", Font.PLAIN, 12);
    metlFont = new Font ("Courier", Font.PLAIN, 12);

    setAllowsChildren (false);

    // I think I don't need v any more
    //parent = v;                              // Sets the parent Vertex
    if (v != null) {          // If not the root operator
        v.setAllowsChildren (true);
        v.add (this);         // Calls DefaultMutableTreeNode's add method
    }
}

//public Vertex getParentVertex ()
//{
//    return parent;
// }

/**
 * Returns the id of this component.
 */
public int getId ()
{
    return id;
}

/**
 * Sets the id of this component to the specified value.
 */
public void setId (int i)
{
    id = i;
}

/**
 * Sets the label of this component to the specified value.
 */
public void setLabel (String s)
{
    label = s;
}

/**
 * Returns the label of this component.
 */
public String getLabel ()
{
    return label;
}

/**
 * Returns the x-component of the offset of the label.
 */
public int getLabelXOffset ()
{
    return labelXOffset;
```

266

```java
    }

    /**
     * Sets the x-component of the offset of the label to the specified
     * value.
     */
    public void setLabelXOffset (int xLoc)
    {
        labelXOffset = xLoc;
    }

    /**
     * Sets the y-component of the offset of the label to the specified
     * value.
     */
    public void setLabelYOffset (int yLoc)
    {
        labelYOffset = yLoc;
    }

    /**
     * Returns the y-component of the offset of the label.
     */
    public int getLabelYOffset ()
    {
        return labelYOffset;
    }

    /**
     * Changes the label offset to the specified x and y values.
     */
    public void setLabelOffset (int xOffset, int yOffset)
    {
        labelXOffset = labelXOffset + xOffset;
        labelYOffset = labelYOffset + yOffset;
    }

    /**
     * Returns font of the label.
     */
    public Font getlFont ()
    {
        return lFont;
    }

    /**
     * Sets the met (or latency) of this component to the specified value.
     */
    public void setMet (PSDLTime s)
    {
        met = s;
    }

    /**
     * Returns the met (or latency) of this component.
     */
    public PSDLTime getMet ()
    {
        return met;
    }

    /**
     * Returns the x-component of the offset of the met (or latency).
     */
    public int getMetXOffset ()
    {
        return metXOffset;
    }

    /**
     * Sets the x-component of the offset of the met (or latency)    to the
     * specified value.
     */
    public void setMetXOffset (int xLoc)
    {
        metXOffset = xLoc;
    }

    /**
     * Sets the y-component of the offset of the met (or latency) to the
     * specified value.
     */
    public void setMetYOffset (int yLoc)
    {
        metYOffset = yLoc;
    }

    /**
     * Returns the y-component of the offset of the met (or latency).
     */
    public int getMetYOffset ()
    {
        return metYOffset;
    }

    /**
     * Changes the met (or latency) offset to the specified x and y
     * values.
     */
    public void setMetOffset (int xOffset, int yOffset)
    {
        metXOffset = metXOffset + xOffset;
        metYOffset = metYOffset + yOffset;
    }
```

```
  /**
   * Returns font of the met (or latency).
   */
  public Font getMetlFont ()
  {
    return metlFont;
  }

  /**
   * This abstract method is implemented in the subclasses.
   */
  public abstract int getX ();

  /**
   * This abstract method is implemented in the subclasses.
   */
  public abstract int getY ();

  /**
   * This abstract method is implemented in the subclasses.
   */
  public abstract void moveTo (int xOffset, int yOffset);

  /**
   * Returns the name (label) of this component.
   */
  public String toString ()
  {
    return label;
  }

  /**
   * Changes the label font index to the specified value.
   */
  public void setLabelFontIndex (int f)
  {
    labelFont = f;
    int type = ((FontConstants.FONT_VALUES [(f - 1) * 3 +
1].equals("Plain"))
                ? Font.PLAIN : Font.BOLD);
    lFont = new Font (FontConstants.FONT_VALUES [(f - 1) * 3],
                      type,
                      Integer.parseInt (FontConstants.FONT_VALUES [(f -
1) * 3 + 2]));
  }

  /**
   * Returns the label font index of this component.
   */
  public int getLabelFontIndex ()
  {
    return labelFont;
  }

  /**
   * Changes the met (or latency) font index to the specified value.
   */
  public void setMetFontIndex (int f)
  {
    metFont = f;
    int type = ((FontConstants.FONT_VALUES [(f - 1) * 3 +
1].equals("Plain"))
                ? Font.PLAIN : Font.BOLD);
    metlFont = new Font (FontConstants.FONT_VALUES [(f - 1) * 3],
                         type,
                         Integer.parseInt (FontConstants.FONT_VALUES [(f -
1) * 3 + 2]));
  }

  /**
   * Returns the met (or latency) font index of this component.
   */
  public int getMetFontIndex ()
  {
    return metFont;
  }

} // End of class DataFlowComponent
```

```java
package caps.Psdl;

import java.util.Vector;
import java.util.Enumeration;
import java.io.*;

public class DataTypes {

    private Vector types;

    private Vector specs;

    private Vector impls;

    public DataTypes ()
    {

        types = new Vector ();
        specs = new Vector ();
        impls = new Vector ();
    }

    // This will be called from the builder
    public void addType (String name, String spec, String impl)
    {

        if (!exists (name)) {
            types.addElement (name);
            specs.addElement (spec);
            impls.addElement (impl);
        }

    }

    // This will be called when a new edge is created
    public void addType (String name)
    {

        if (!exists (name) && !isPredefined (name)) {
            types.addElement (name);
            specs.addElement ("\nEND");
            impls.addElement ("ada " + name + "\nEND");
        }

    }

    public boolean exists (String name)
    {

        boolean flag = false;
        for (Enumeration enum = types.elements (); enum.hasMoreElements ();)
        {

            if (name.equals((String) enum.nextElement ()))
                flag = true;
        }
        return flag;

    }

    public boolean isPredefined (String str)
    {

        if (str.equalsIgnoreCase ("boolean") || str.equalsIgnoreCase
("character") ||
            str.equalsIgnoreCase ("string") || str.equalsIgnoreCase
("integer") ||
            str.equalsIgnoreCase ("real") || str.equalsIgnoreCase
("exception"))
            return true;
        else
            return false;

    }

    // this is called when reafing form the file for the first time
    public void buildTypes (File file)
    {

        StreamTokenizer tok = null;
        try {
            tok = new StreamTokenizer (new FileReader (file));
        } catch (FileNotFoundException ex) {
            System.out.println (ex);
        }
        build (tok);

    }

    // called when building types in the editor
    public void buildTypes (String s)
    {

        StreamTokenizer tok = null;
        tok = new StreamTokenizer (new StringReader (s));
        build (tok);

    }

    private void build (StreamTokenizer tok)
    {

        removeElements ();
        String str;
        String tempStr;
        tok.wordChars (33, 126);
        tok.eolIsSignificant (true);
        int tokType;
        int counter = 0;

        try {
            while ((tokType = tok.nextToken ()) != = StreamTokenizer.TT_EOF)
                if (tokType == StreamTokenizer.TT_WORD &&
tok.sval.equalsIgnoreCase ("TYPE")) {
                    tempStr = getNextToken (tok);
                    str = tempStr;
                    types.addElement (str);
                    do {
                        tempStr = getNextToken (tok);
```

```java
        } while (!tempStr.equalsIgnoreCase ("SPECIFICATION"));
        str = "";
        counter++;
        do {
            tempStr = getNextToken (tok);
            if (tempStr.equalsIgnoreCase ("SPECIFICATION"))
                counter++;
            else if (tempStr.equalsIgnoreCase ("END"))
                counter--;
            str = str.concat (tempStr);
            if (tempStr != "\n")
                str = str.concat (" ");
        } while (counter > 0 || !tempStr.equalsIgnoreCase ("END"));
        specs.addElement (str);
        do {
            tempStr = getNextToken (tok);
        } while (!tempStr.equalsIgnoreCase ("IMPLEMENTATION"));
        str = "";
        counter++;
        do {
            tempStr = getNextToken (tok);
            if (tempStr.equalsIgnoreCase ("IMPLEMENTATION"))
                counter++;
            else if (tempStr.equalsIgnoreCase ("END"))
                counter--;
            str = str.concat (tempStr);
            if (tempStr != "\n")
                str = str.concat (" ");
        } while (counter != 0 || !tempStr.equalsIgnoreCase ("END"));
        impls.addElement (str);
    }

    } catch (IOException ex) {
        System.out.println (ex);
    }
    }
}

public String getNextToken (StreamTokenizer tok) throws IOException
{
    String str = "";
    tok.nextToken ();
    if (tok.ttype == StreamTokenizer.TT_EOL)
        str = "\n";
    else if (tok.ttype == StreamTokenizer.TT_WORD)
        str = str.concat (tok.sval);
    return str;
}

public void removeElements ()
{
    types.removeAllElements ();
    specs.removeAllElements ();
    impls.removeAllElements ();
}

public String toString ()
{
    String str = "";
    int numberOfTypes = types.size ();
    for (int ix = 0; ix < numberOfTypes; ix++) {
        str = str.concat ("TYPE " + (String) types.elementAt (ix) + "\n");
        str = str.concat ("SPECIFICATION " + (String) specs.elementAt (ix) + "\n");
        str = str.concat ("IMPLEMENTATION " + (String) impls.elementAt (ix) + "\n\n");
    }
    return str;
}

} // End of the class DataTypes.
```

```java
package caps.Psdl;

import java.util.Vector;
import java.util.Enumeration;
import java.awt.Point;
import java.io.StringReader;
import java.io.StreamTokenizer;
import java.io.IOException;

/**
 * Edge represents a stream in the data flow diagram
 * It is also a TreeNode object
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class Edge extends DataFlowComponent {

    /**
     * The source Vertex of this stream.
     */
    protected Vertex source;

    /**
     * The destination Vertex of this stream.
     */
    protected Vertex destination;

    /**
     * The vector that holds the control points of this stream.
     */
    protected Vector points;

    /**
     * The type name of the stream.
     */
    protected String streamType;

    /**
     * The initial value of the stream.
     */
    protected String initialValue;

    /**
     * True if this is a state stream.
     */
    protected boolean isState;

    /**
     * The x location of this stream in the DrawPanel.
     */
    protected int x;

    /**
     * The y location of this stream in the DrawPanel.
     */
    protected int y;

    /**
     * The index of the handle that the mouse is pressed on.
     */
    protected int selectedHandleIndex;

    /**
     * The constructor for this class.
     *
     * @param v the parent vertex of this edge.
     */
    public Edge (int xLocation, int yLocation, Vertex v)
    {
        super (v);
        source = null;
        destination = null;
        points = new Vector (0, 2);
        points.addElement (new Point (xLocation, yLocation));
        streamType = "undefined_type";
        initialValue = "";
        setLabel ("unnamed_stream_" + UNIQUE_ID++);

        id = ++UNIQUE_ID;

        isState = false;
        met = null;
        setX (xLocation);
        setY (yLocation);
    }

    /**
     * Relocates the stream when the stream is moved with other objects.
     */
    public void moveTo (int xOffset, int yOffset)
    {
        Point p;
        for (Enumeration enum = points.elements (); enum.hasMoreElements
();) {
            p = (Point) enum.nextElement ();
            p.x = p.x + xOffset;
            p.y = p.y + yOffset;
        }
        correctLabelOffset ();
    }

    /**
```

271

```java
    * Is called when one of the handles of the stream is dragged in the
DrawPanel.
    */
public void reShape (int xLocation, int yLocation)
{
    if (selectedHandleIndex == 0) {        // the source
        return;
    }

    if (points.size () == 3) {             // has only one control point,
add more
        Point begin = (Point) points.elementAt (0);
        Point end = (Point) points.elementAt (2);
        int xDiff = (end.x - begin.x) / 6;
        int yDiff = (end.y - begin.y) / 6;
        points.removeElementAt (1);
        for (int index = 1; index < 6; index++) {
            points.add (index, new Point (begin.x + xDiff * index, begin.y
+ yDiff * index));
        }

        selectedHandleIndex = 3;
    }

    Point p = (Point) points.elementAt (selectedHandleIndex);
    Point prev = (Point) points.elementAt (selectedHandleIndex - 1);
    Point next = (Point) points.elementAt (selectedHandleIndex + 1);
    Point middle;

    int diffX = xLocation - p.x;
    int diffY = yLocation - p.y;
    p.x = p.x + diffX;
    p.y = p.y + diffY;

    if (selectedHandleIndex == 1) {
        next.x = next.x + diffX * 2;
        next.y = next.y + diffY * 2;

        Point nextControl= (Point) points.elementAt (selectedHandleIndex
+ 3);
        middle = (Point) points.elementAt (selectedHandleIndex + 2);
        middle.x = (next.x + nextControl.x) / 2;
        middle.y = (next.y + nextControl.y) / 2;
    }
    else if (selectedHandleIndex == points.size () - 2) {
        prev.x = prev.x + diffX * 2;
        prev.y = prev.y + diffY * 2;

        Point prevControl= (Point) points.elementAt (selectedHandleIndex
- 3);
        middle = (Point) points.elementAt (selectedHandleIndex - 2);
        middle.x = (prev.x + prevControl.x) / 2;
        middle.y = (prev.y + prevControl.y) / 2;
    }
    else {
        prev.x = prev.x + diffX;
        prev.y = prev.y + diffY;
        Point prevControl= (Point) points.elementAt (selectedHandleIndex
- 3);
        middle = (Point) points.elementAt (selectedHandleIndex - 2);
        middle.x = (prev.x + prevControl.x) / 2;
        middle.y = (prev.y + prevControl.y) / 2;

        next.x = next.x + diffX;
        next.y = next.y + diffY;
        Point nextControl= (Point) points.elementAt (selectedHandleIndex
+ 3);
        middle = (Point) points.elementAt (selectedHandleIndex + 2);
        middle.x = (next.x + nextControl.x) / 2;
        middle.y = (next.y + nextControl.y) / 2;
    }

    correctLabelOffset ();
}

/**
 * Changes the x value of the stream to the specified value.
 */
public void setX (int newX)
{
    x = newX;
}

/**
 * Changes the y value of the stream to the specified value.
 */
public void setY (int newY)
{
    y = newY;
}

/**
 * Changes selectedHandleIndex to the specified value.
 */
public void setSelectedHandleIndex (int i)
{
    selectedHandleIndex = i;
}

/**
 * Returns the x value of this stream.
 */
public int getX ()
{
    return x;    // **** Pending ****
}
```

```java
/**
 * Returns the y value of this stream.
 */
public int getY ()
{
    return y; // **** Pending ****
}

/**
 * Returns the source Vertex of this stream.
 */
public Vertex getSource ()
{
    return source;
}

/**
 * Sets the source Vertex of this stream to the specified value.
 */
public void setSource (Vertex v)
{
    source = v;
}

/**
 * Returns the destination Vertex of this stream.
 */
public Vertex getDestination ()
{
    return destination;
}

/**
 * Sets the destination Vertex of this stream to the specified value.
 */
public void setDestination (Vertex v)
{
    destination = v;
}

/**
 * Returns the type of this stream.
 */
public String getStreamType ()
{
    return streamType;
}

/**
 * Sets the type of this stream to the specified value.
 */
public void setStreamType (String type)
{
    streamType = type;
}

/**
 * Returns true if this is a state stream.
 */
public boolean isStateStream ()
{
    return isState;
}

/**
 * Changes the isState field to the specified value.
 */
public void setStateStream (boolean flag)
{
    isState = flag;
}

/**
 * Returns the initial value of this stream.
 */
public String getInitialValue ()
{
    return initialValue;
}

/**
 * Sets the initial value of this stream to the specified value.
 */
public void setInitialValue (String str)
{
    initialValue = str;
}

/**
 * Adds a new point to the control points. Also adds the middle point
 * of the control points.
 *
 * @param x the x component of the new conrol point.
 * @param y the y component of the new conrol point.
 */
public void addPoint (int x, int y)
{
    Point p = (Point) points.lastElement (); // the last element
    Point middle = new Point ((x + p.x) / 2, (y + p.y) / 2);
    points.addElement (middle);
    points.addElement (new Point (x, y));
}

/**
```

273

```java
 * Returns the control points vector.
 */
public Vector getPoints ()
{
    return points;
}

/**
 * Sets the location of this stream to the middle control point.
 */
// Pending this is called by so many methods needlessly
public void correctLabelOffset ()
{
    Point p = (Point) points.elementAt (points.size () / 2);     // The
middle point in the vector
    setX (p.x + 10); setY (p.y - 10);
}

/**
 * Locates the ending points of this stream on the perimeter of the
source and destination.
 */
public void correctEndingPoints ()
{
    Point p1 = source.getIntersectionPoint ((Point) points.elementAt
(1));    // ikinci ve sondan ikinci elemanlar
    Point p2 = destination.getIntersectionPoint ((Point)
points.elementAt (points.size () - 2));
    Point p3;
    Point middle;

    points.setElementAt (p1, 0);
    p3 = (Point) points.elementAt (2);
    middle = (Point) points.elementAt (1);
    middle.setLocation ((p1.x + p3.x) / 2, (p1.y + p3.y) / 2);

    points.setElementAt (p2, points.size () - 1);
    p3 = (Point) points.elementAt (points.size () - 3);
    middle = (Point) points.elementAt (points.size () - 2);
    middle.setLocation ((p2.x + p3.x) / 2, (p2.y + p3.y) / 2);
    correctLabelOffset ();
}

/**
 * Called to extract a string representation of the control points.
 * Constructs the points vector from the string expression.
 */
// called to build the points vector
public void setInitialControlPoints (String exp)
{
    points.removeAllElements ();

    exp = exp.substring (1, exp.length () - 1);
    exp.trim ();
    StringReader reader = new StringReader (exp);
    StreamTokenizer tok = new StreamTokenizer (reader);
    int tokType;

    if (source instanceof External) {
        try {
            tok.nextToken ();
            source.setX ((int) tok.nval);
            tok.nextToken ();
            source.setY ((int) tok.nval);
        } catch (IOException ex) {
            System.out.println (ex);
        }
    }

    points.addElement (new Point (source.getX (), source.getY ()));

    try {
        while ((tokType = tok.nextToken ()) != StreamTokenizer.TT_EOF) {
            int x = (int) tok.nval;
            tok.nextToken ();
            int y = (int) tok.nval;
            addPoint (x, y);
        }
    } catch (IOException ex) {
        System.out.println (ex);
    }

    if (destination instanceof External) {
        destination.setX (((Point) points.lastElement ()).x);
        destination.setY (((Point) points.lastElement ()).y);
    }
    else
        addPoint (destination.getX (), destination.getY ());
    correctEndingPoints ();
}

/**
 * Deletes this stream.
 */
public void delete (boolean deletingInEdge)
{
    if (deletingInEdge) {
        source.removeOutEdge (this);
    }
    else {
        destination.removeInEdge (this);
    }
    deleteHelper ();
}
```

```java
/**
 * Deletes this stream.
 */
public void delete ()
{
    source.removeOutEdge (this);
    destination.removeInEdge (this);
    deleteHelper ();
}

/**
 * Helper method to delete the stream.
 */
public void deleteHelper ()
{
    if (isStateStream ()) {
        int index = -1;
        label :
        for (Enumeration enum = parent.children (); enum.hasMoreElements
());) {  // trying to find index
            DataFlowComponent dfc = (DataFlowComponent) enum.nextElement
();   // of this child in the
            if (dfc instanceof Edge && ((Edge) dfc).isStateStream ())
// parent's children
                index++;
                if (dfc.equals (this))
                    break label;
        }
        ((Vector) ((Vertex) parent).getSpecReqmts ()).elementAt
(2)).removeElementAt (index);
    }
    if (source instanceof External)
        source.delete ();
    if (destination instanceof External)
        destination.delete ();
    removeFromParent ();
}

} // End of the class Edge.
```

```java
package caps.Psdl;

import java.awt.Point;

public class External extends Vertex {

    public External (int xLocation, int yLocation, Vertex v)
    {
        super (xLocation, yLocation, v, false);

        met = null;
        setLabel ("EXTERNAL");

        labelYOffset = 10;

        x = xLocation;
        y = yLocation;
        width = 0;
        height = 0;

        removeFromParent ();
    }

    public Point getIntersectionPoint (Point p)
    {
        return new Point (x, y);
    }

} // End of the class External
```

```java
package caps.Psdl;

/**
 * This class represents a combination of time value from an integer that
represents
 * the time and another integer that represents the unit.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class PSDLTime extends Object {

   /**
    * The constant value for microseconds.
    */
   public final static int microsec = 0;

   /**
    * The constant value for miliseconds.
    */
   public final static int ms = 1;

   /**
    * The constant value for seconds.
    */
   public final static int sec = 2;

   /**
    * The constant value for minutes.
    */
   public final static int min = 3;

   /**
    * The constant value for hours.
    */
   public final static int hours = 4;

   /**
    * The value of the time.
    */
   private int value;

   /**
    * The units of the time.
    */
   private int units;

   /**
    * The constructor for this class.
    */
   public PSDLTime() {
      value = 0;
      units = ms;
   }

   /**
    * The constructor for this class.
    *
    * @param timeValue the value of the time.
    * @param timeUnits the unit of the time.
    */
   public PSDLTime(int timeValue, int timeUnits) {
      value = timeValue;
      units = timeUnits;
   }

   /**
    * Returns the time time value of this object.
    */
   public int getTimeValue() {
      return value;
   }

   /**
    * Sets the time value to the specified argument.
    */
   public void setTimeValue(int timeValue) {
      value = timeValue;
   }

   /**
    * Returns the time units of this object.
    */
   public int getTimeUnits() {
      return units;
   }

   /**
    * Sets the time unit to the specified argument.
    */
   public void setTimeUnits(int timeUnits) {
      units = timeUnits;
   }

   /**
    * Sets the time unit to the specified argument.
    */
   public void setTimeUnits (String u)
   {
      if (u == "microsec")
         units = microsec;
      else if (u == "ms")
         units = ms;
      else if (u == "sec")
```

```java
package caps.Psdl;

import java.awt.Font;
import java.util.*;
import java.awt.Point;
import java.io.BufferedReader;
import java.io.StringReader;
import java.io.IOException;

/**
 * This class represents a terminator or an operator.
 * It holds the data structures that represent the constructs for the
Vertex.
 *
 * @author Ilker DURANLIOGLU
 * @version
 */
public class Vertex extends DataFlowComponent {

    /**
     * The constant value for the intitial radius of a Vertex.
     */
    public static final int INITIAL_RADIUS = 35;

    /**
     * The constant value for non-time ciritical Vertices.
     */
    public static final int NON_TIME_CRITICAL = 0;

    /**
     * The constant value for periodic Vertices.
     */
    public static final int PERIODIC = 1;

    /**
     * The constant value for sporadic Vertices.
     */
    public static final int SPORADIC = 2;

    /**
     * The constant value for unprotected Vertices.
     */
    public static final int UNPROTECTED = 0;

    /**
     * The constant value for Vertices that have "BY SOME" triggering
construct.
     */
    public static final int BY_SOME = 1;

    /**
```

```java
        units = sec;
    else if (u == "min")
        units = min;
    else if (u == "hours")
        units = hours;
    }

    /**
     * Returns a string representation of this object.
     *
     * @return the string representation in the form of "12 sec"
     */
    public String toString() {
        String unitString;
        switch (units) {
            case microsec : unitString = "microsec";
                break;
            case ms : unitString = "ms";
                break;
            case sec : unitString = "sec";
                break;
            case min : unitString = "min";
                break;
            case hours : unitString = "hours";
                break;
            default : unitString = "undefined";
        }
        return String.valueOf(value) + " " + unitString;
    }

} // End of the class PSDLTime
```

```java
 * The constant value for Vertices that have "BY ALL" triggering
construct.
 */
public static final int BY_ALL = 2;

/**
 * True if this Vertex is a terminator.
 */
protected boolean terminator;

/**
 * The x-location of this component on the DrawPanel
 */
protected int x;

/**
 * The y-location of this component on the DrawPanel
 */
protected int y;

/**
 * The width of this component.
 * It serves as the radius of an operator and the width of a
terminator width of operator .cap
 */
protected int width;

/**
 * The height of this component.
 */
protected int height;

/**
 * The color parameter of this component.
 */
protected int color;

protected Vector metReqmts;
protected PSDLTime period;
protected Vector periodReqmts;
protected PSDLTime finishWithin;
protected Vector finishWithinReqmts;
protected PSDLTime mcp;
protected Vector mcpReqmts;
protected PSDLTime mrt;

protected Vector mrtReqmts;

protected int timingType;

protected int triggerType;

protected Vector triggerReqmts;

protected Vector triggerStreamsList;

protected String ifCondition;

protected String outputGuardList;

protected String exceptionGuardList;

protected String exceptionList;

protected String timerOpList;

protected Vector keywordList;

protected String informalDesc;

protected String formalDesc;

protected Vector inEdges;

protected Vector outEdges;

protected String impLanguage;

protected Vector timerList;

protected String graphDesc;

protected String genericList;

protected Vector specReqmts;

/**
 * The constructor for this class.
 *
 * @param xlocation The x component of the location of this component.
 * @param ylocation The y component of the location of this component.
 * @param v         The parent vertex of this component.
 * @param t         true if this component is a terminator.
 */
public Vertex (int xLocation, int yLocation, Vertex v, boolean t)
{
    super (v);
```

```
inEdges = new Vector (0);
outEdges = new Vector (0);

terminator = t;

color = 62;  // initially white

timingType = NON_TIME_CRITICAL;
triggerType = UNPROTECTED;

met = null;

if (v == null) {
    setLabel ("root_" + UNIQUE_ID++);
}
else if (isTerminator ()) {
    setLabel ("terminator_" + UNIQUE_ID++);
    met = new PSDLTime ();
}
else {
    setLabel ("operator_" + UNIQUE_ID++);
}

setWidth (INITIAL_RADIUS * 2);

if (getParent () == null)          //if this is the root
    id = ++UNIQUE_ID;
else {
    UNIQUE_ID++;          // op_num
    id = ++UNIQUE_ID;
}

period = null;
finishWithin = null;
mcp = null;
mrt = null;
metReqmts = new Vector (0, 2);
periodReqmts = new Vector (0, 2);
finishWithinReqmts = new Vector (0, 2);
mcpReqmts = new Vector (0, 2);
mrtReqmts = new Vector (0, 2);
triggerReqmts = new Vector (0, 2);
triggerStreamsList = new Vector (0, 2);
ifCondition = "";
outputGuardList = "";
exceptionGuardList = "";
exceptionList = "";
timerOpList = "";
keywordList = new Vector (0, 2);
informalDesc = "";
formalDesc = "";

timerList = new Vector (0, 2);
graphDesc = "";
genericList = "";
specReqmts = new Vector (0, 2);
specReqmts.addElement (new Vector (0, 2));
specReqmts.addElement (new Vector (0, 2));
specReqmts.addElement (new Vector (0, 2));

impLanguage = "ada";

x = xLocation;
y = yLocation; // Set the location of the component
}

/**
 * Sets the location of this component on the screen.
 * Also corrects the location of the ending streams.
 *
 * @param xLocation The new x component of the location on the
drawpanel
 * @param yLocation The new y component of the location on the
drawpanel
 */
public void setLocation (int xOffset, int yOffset)
{
    moveTo (xOffset, yOffset);
    correctInOutStreams ();
}

/**
 * Sets the location of this component on the screen.
 *
 * @param xLocation The new x component of the location on the
drawpanel
 * @param yLocation The new y component of the location on the
drawpanel
 */
public void moveTo (int xOffset, int yOffset)
{
    x = x + xOffset;
    y = y + yOffset;
}

/**
 * Returns true if this component is a terminator.
 *
 * @return true if this component is a terminator.
 */
public boolean isTerminator ()
{
    return terminator;
}
```

```java
/**
 * Sets this component as a terminator or a stream.
 * Also changes the width of the component.
 *
 * @param b.
 */
public void setTerminator (boolean b)
{
    terminator = b;
    setWidth (width);
}

/**
 * Corrects the ending points of the in and out streams of this
component.
 */
public void correctInOutStreams ()
{
    for (Enumeration enum = inEdges.elements (); enum.hasMoreElements
();) {
        ((Edge) enum.nextElement ()).correctEndingPoints ();
    for (Enumeration enum = outEdges.elements (); enum.hasMoreElements
();) {
        ((Edge) enum.nextElement ()).correctEndingPoints ();
    }
}

/**
 * Returns the width of this component.
 *
 * @return the width of this component.
 */
public int getWidth ()
{
    return width;
}

/**
 * Changes the width of this component.
 *
 * @param w the new width of this component.
 */
public void setWidth (int w)
{
    width = w;
    if (isTerminator ())
        height = (int) (width / 1.4d);
    else
        height = width;
}

/**
 * Returns the height of this component.
 *
 * @return the height of this component.
 */
public int getHeight ()
{
    return height;
}

/**
 * Returns the x component of the location of this Vertex
 *
 * @return x
 */
public int getX ()
{
    return x;
}

/**
 * Changes the x component of the location of this Vertex
 *
 * @param xLoc.
 */
public void setX (int xLoc)
{
    x = xLoc;
}

/**
 * Changes the y component of the location of this Vertex.
 *
 * @param yLoc.
 */
public void setY (int yLoc)
{
    y = yLoc;
}

/**
 * Returns the y component of the location of this Vertex.
 *
 * @return y
 */
public int getY ()
{
    return y;
}

/**
```

```java
 * Changes the color value for this Vertex.
 *
 * @param c the new color value.
 */
public void setColor (int c)
{
    color = c;
}

/**
 * Returns the color value for this Vertex.
 *
 * @return the color value of the Vertex.
 */
public int getColor ()
{
    return color;
}

/**
 * Adds a new Edge to the inEdges Vector.
 *
 * @param e the new inEdge.
 */
public void addInEdge (Edge e)
{
    inEdges.addElement (e);
    ((Vector) specReqmts.elementAt (0)).addElement ("");
}

/**
 * Removes an Edge from the inEdges Vector.
 *
 * @param e the inEdge to be removed.
 */
public void removeInEdge (Edge e)
{
    int index = inEdges.indexOf (e);
    inEdges.removeElementAt (index);
    ((Vector) specReqmts.elementAt (0)).removeElementAt (index);
}

/**
 * Adds a new Edge to the outEdges Vector.
 *
 * @param e the new outEdge.
 */
public void addOutEdge (Edge e)
{
    outEdges.addElement (e);
    ((Vector) specReqmts.elementAt (1)).addElement ("");
}

/**
 * Removes an Edge from the outEdges Vector.
 *
 * @param e the outEdge to be removed.
 */
public void removeOutEdge (Edge e)
{
    int index = outEdges.indexOf (e);
    outEdges.removeElementAt (index);
    ((Vector) specReqmts.elementAt (1)).removeElementAt (index);
}

/**
 * Returns the timing type of this Vertex.
 */
public int getTimingType ()
{
    return timingType;
}

/**
 * Sets the timing type to the specified value.
 */
public void setTimingType (int type)
{
    timingType = type;
}

/**
 * Returns the triggering type of this Vertex.
 */
public int getTriggerType ()
{
    return triggerType;
}

/**
 * Sets the triggering type to the specified value.
 */
public void setTriggerType (int type)        // I should throw some
exceptions here
{
    triggerType = type;
}

/**
 * Returns the period value of this Vertex.
 */
public PSDLTime getPeriod ()
{
    return period;
```

```java
}

/**
 * Returns the finish within value of this Vertex.
 */
public PSDLTime getFinishWithin ()
{
    return finishWithin;
}

/**
 * Returns the mcp value of this Vertex.
 */
public PSDLTime getMcp ()
{
    return mcp;
}

/**
 * Returns the mrt value of this Vertex.
 */
public PSDLTime getMrt ()
{
    return mrt;
}

/**
 * Sets the period to the specified value.
 */
public void setPeriod (PSDLTime p)
{
    period = p;
}

/**
 * Sets the finish within to the specified value.
 */
public void setFinishWithin (PSDLTime fw)
{
    finishWithin = fw;
}

/**
 * Sets the mcp to the specified value.
 */
public void setMcp (PSDLTime m)
{
    mcp = m;
}

/**
 * Sets the mrt to the specified value.
 */
public void setMrt (PSDLTime mr)
{
    mrt = mr;
}

/**
 * Returns the implementation language of this Vertex.
 */
public String getImpLanguage ()
{
    return impLanguage;
}

/**
 * Sets the implementation language to the specified value.
 */
public void setImpLanguage (String s)
{
    impLanguage = s;
}

/**
 * Returns the met requirements of this Vertex.
 */
public Vector getMetReqmts ()
{
    return metReqmts;
}

/**
 * Sets the met requirements to the specified value.
 */
public void setMetReqmts (Vector v)
{
    metReqmts = v;
}

/**
 * Returns the period requirements of this Vertex.
 */
public Vector getPeriodReqmts ()
{
    return periodReqmts;
}

/**
 * Sets the period requirements to the specified value.
 */
public void setPeriodReqmts (Vector v)
{
    periodReqmts = v;
```

```java
}

/**
 * Returns the finish within requirements of this Vertex.
 */
public Vector getFinishWithinReqmts ()
{
    return finishWithinReqmts;
}

/**
 * Sets the finish within requirements to the specified value.
 */
public void setFinishWithinReqmts (Vector v)
{
    finishWithinReqmts = v;
}

/**
 * Returns the mcp requirements of this Vertex.
 */
public Vector getMcpReqmts ()
{
    return mcpReqmts;
}

/**
 * Sets the mcp requirements to the specified value.
 */
public void setMcpReqmts (Vector v)
{
    mcpReqmts = v;
}

/**
 * Returns the mrt requirements of this Vertex.
 */
public Vector getMrtReqmts ()
{
    return mrtReqmts;
}

/**
 * Sets the mrt requirements to the specified value.
 */
public void setMrtReqmts (Vector v)
{
    mrtReqmts = v;
}

/**
 * Returns the trigger requirements of this Vertex.
 */
public Vector getTriggerReqmts ()
{
    return triggerReqmts;
}

/**
 * Sets the trigger requirements to the specified value.
 */
public void setTriggerReqmts (Vector v)
{
    triggerReqmts = v;
}

/**
 * Returns the triggering streams of this Vertex.
 */
public Vector getTriggerStreamsList ()
{
    return triggerStreamsList;
}

/**
 * Sets the trigger streams list to the specified value.
 */
public void setTriggerStreamsList (Vector v)
{
    triggerStreamsList = v;
}

/**
 * Returns the if condition of this Vertex.
 */
public String getIfCondition ()
{
    return ifCondition;
}

/**
 * Sets the if condition to the specified value.
 */
public void setIfCondition (String s)
{
    ifCondition = s;
}

/**
 * Returns the output guard list of this Vertex.
 */
public String getOutputGuardList ()
{
    return outputGuardList;
```

```java
    }

    /**
     * Sets the output guard list to the specified value.
     */
    public void setOutputGuardList (String s)
    {
        outputGuardList = s;
    }

    /**
     * Returns the exception guard list of this Vertex.
     */
    public String getExceptionGuardList ()
    {
        return exceptionGuardList;
    }

    /**
     * Sets the exception guards list to the specified value.
     */
    public void setExceptionGuardList (String s)
    {
        exceptionGuardList = s;
    }

    /**
     * Returns the exception list of this Vertex.
     */
    public String getExceptionList ()
    {
        return exceptionList;
    }

    /**
     * Sets the exception list to the specified value.
     */
    public void setExceptionList (String s)
    {
        exceptionList = s;
    }

    /**
     * Returns the timer op list of this Vertex.
     */
    public String getTimerOpList ()
    {
        return timerOpList;
    }

    /**
     * Sets the timer op list to the specified value.
     */
    public void setTimerOpList (String s)
    {
        timerOpList = s;
    }

    /**
     * Returns the informal description of this Vertex.
     */
    public String getInformalDesc ()
    {
        return informalDesc;
    }

    /**
     * Sets the informal description to the specified value.
     */
    public void setInformalDesc (String s)
    {
        informalDesc = s;
    }

    /**
     * Returns the formal description of this Vertex.
     */
    public String getFormalDesc ()
    {
        return formalDesc;
    }

    /**
     * Sets the formal description to the specified value.
     */
    public void setFormalDesc (String s)
    {
        formalDesc = s;
    }

    /**
     * Returns the keywords of this Vertex.
     */
    public Vector getKeywordList ()
    {
        return keywordList;
    }

    /**
     * Sets the keywords to the specified value.
     */
    public void setKeywordList (Vector v)
    {
        keywordList = v;
```

```java
    }
    /**
     * Returns the timers of this Vertex.
     */
    public Vector getTimerList ()
    {
        return timerList;
    }
    /**
     * Sets the timer list to the specified value.
     */
    public void setTimerList (Vector v)
    {
        timerList = v;
    }
    /**
     * Returns the informal graph description of this Vertex.
     */
    public String getGraphDesc ()
    {
        return graphDesc;
    }
    /**
     * Sets the graph description to the specified value.
     */
    public void setGraphDesc (String s)
    {
        graphDesc = s;
    }
    /**
     * Returns the generic list of this Vertex.
     */
    public String getGenericList ()
    {
        return genericList;
    }
    /**
     * Sets the generic list to the specified value.
     */
    public void setGenericList (String s)
    {
        genericList = s;
    }
    /**
     * Returns the spec requirements of this Vertex.
     */
    public Vector getSpecReqmts ()
    {
        return specReqmts;
    }
    /**
     * Returns intersection point of this vertex with the specified point.
     */
    public Point getIntersectionPoint (Point p)
    {
        if (isTerminator ())
            return getTerminatorIntersection (p);
        else
            return getOperatorIntersection (p);
    }
    /**
     * Returns the intersection point of this vertex with the specified
point.
     * Called from getIntersectionPoint when this Vertex is a Terminator
     */
    public Point getTerminatorIntersection (Point p)
    {
        int x;
        int y;
        float slope;

        slope = (float) (p.y - getY ()) / (float) (p.x - getX ());
        if (Math.abs (slope) >= (1 / 1.4f) {
            if (p.y <= getY () - getHeight () / 2)
                y = getY () - getHeight () / 2;
            else
                y = getY () + getHeight () / 2;
            x = (int) ((float) (y - getY ()) / slope) + getX ();
        }
        else {
            if (p.x <= getX () - getWidth () / 2)
                x = getX () - getWidth () / 2;
            else
                x = getX () + getWidth () / 2;
            y = (int) ((float) (x - getX ()) * slope) + getY ();
        }
        return new Point (x, y);
    }
    /**
     * Returns the intersection point of this vertex with the specified
point.
     * Called from getIntersectionPoint when this Vertex is an Operator.
     */
    public Point getOperatorIntersection (Point p)
```

```java
{
    double distance = p.distance (getX (), getY ());    // Distance from
the point to the center
    int x = getX () + (int) (((double) (getWidth () / 2) / distance)
                             * (float) (p.x - getX ()));
    int y = getY () + (int) (((double) (getWidth () / 2) / distance)
                             * (float) (p.y - getY ()));

    return new Point (x, y);
}

/**
 * Creates the specification construct from its data structures.
 *
 * @param hasId boolean value that specifies if this Vertex has a
unique id.
 * @return returns the string representation of the specification of
this Vertex.
 */
public String getSpecification (boolean hasId)
{
    Enumeration en;
    String tmp;
    String spec = "";
    try {
        if (hasId)
            spec = spec.concat ("OPERATOR " + getLabel () + "_" + getId () +
"\n");
        else
            spec = spec.concat ("OPERATOR " + getLabel () + "\n");
        spec = spec.concat ("    SPECIFICATION\n");
        if (genericList.length () != 0 )
            spec = spec.concat (genericList + "\n");
        en = ((Vector) specReqmts.elementAt (0)).elements ();
        String input = "";    // Because otherwise it allows more
than one input or output******
        for (Enumeration enum = inEdges.elements (); enum.hasMoreElements
();) {
            Edge e = (Edge) enum.nextElement ();
            if (input.lastIndexOf (" " + e.getLabel () + " ") == -1) {
                input = input.concat ("    INPUT " + e.getLabel () + " : " +
e.getStreamType () + "\n");
                if ((tmp = (String) en.nextElement ()).length () != 0)
                    input = input.concat ("    REQUIRED BY " + tmp +
"\n");
            }
        }
        spec = spec.concat (input);
        en = ((Vector) specReqmts.elementAt (1)).elements ();
        String output = "";
        for (Enumeration enum = outEdges.elements (); enum.hasMoreElements
();) {
            Edge e = (Edge) enum.nextElement ();
            if (output.lastIndexOf (" " + e.getLabel () + " ") == -1) {
                output = output.concat ("    OUTPUT " + e.getLabel () + " : "
" + e.getStreamType () + "\n");
                if ((tmp = (String) en.nextElement ()).length () != 0)
                    output = output.concat ("    REQUIRED BY " + tmp +
"\n");
            }
        }
        spec = spec.concat (output);
        en = ((Vector) specReqmts.elementAt (2)).elements ();
        String state = "";
        for (Enumeration enum = children (); enum.hasMoreElements ();) {
            DataFlowComponent d = (DataFlowComponent) enum.nextElement ();
            if (d instanceof Edge && ((Edge) d).isStateStream ()) {
                if (state.lastIndexOf (" " + ((Edge) d).getLabel () + " ") ==
-1) {
                    state = state.concat ("    STATES " + ((Edge) d).getLabel
() + " : "
                                          + ((Edge) d).getStreamType () + "
INITIALLY "
                                          + ((Edge) d).getInitialValue () +
"\n");
                }
            }
            if ((tmp = (String) en.nextElement ()).length () != 0)
                state = state.concat ("    REQUIRED BY " + tmp +
"\n");
        }
        spec = spec.concat (state);
        if (exceptionList.length () != 0)
            spec = spec.concat (extractString (exceptionList, false));
        if (met != null) {
            spec = spec.concat ("    MAXIMUM EXECUTION TIME " +
met.toString () + "\n");
            if (!metReqmts.isEmpty ())
                spec = spec.concat ("    REQUIRED BY " + extractList
(metReqmts) + "\n");
        }
        if (keywordList.size () > 0)
            spec = spec.concat ("    KEYWORDS " + extractList (keywordList)
+ "\n");
        if (informalDesc.length () != 0)
            spec = spec.concat (extractString (informalDesc, false));
        if (formalDesc.length () != 0)
            spec = spec.concat (extractString (formalDesc, false));
        spec = spec.concat ("    END");
    } catch (Exception ex) {
        System.out.println (spec + "\n" + ex);
    }
    return spec;
}
```

```java
/**
 * Called from getSpecification.
 * Extracts the string parameter and reformats it to add to the
specification.
 */
public String extractString (String str, boolean moreSpaces)
{
    BufferedReader reader = new BufferedReader (new StringReader (str));
    str = "";
    String line = "";
    try {
        while ((line = reader.readLine ()) != null) {
            if (moreSpaces)              // from exceptionGuardList,
outputGuardList and timerOpList
                str = str.concat ("          " + line + "\n");
            else                        // from exceptions formal and informal
description
                str = str.concat ("     " + line + "\n");
        }
    } catch (IOException ex) {
        System.out.println (ex);
    }
    return str;
}

/**
 * Extracts an idList which is represented as a Vector and returns a
String representation
 * of the idList so that it will have the form "id1, id2, id3..."
 */
public String extractList (Vector v)
{
    String str = "";
    Enumeration enum;
    if (v != null) {
        enum = v.elements ();
        if (enum.hasMoreElements ())
            str = new String ((String) enum.nextElement ());
        while (enum.hasMoreElements ()) {
            str = str.concat (", ").concat ((String) enum.nextElement ());
        }
    }
    return str;
}

/**
 * Deletes this Vertex.
 * Deletes all the children of this Vertex and also deletes all the in
and out Edges.
 */
public void delete ()
{
    for (Enumeration enum = children (); enum.hasMoreElements ();) {
        DataFlowComponent dfc = (DataFlowComponent) enum.nextElement ();
        if (dfc instanceof Vertex)
            ((Vertex) dfc).delete ();
    }
    for (Enumeration enum = inEdges.elements (); enum.hasMoreElements
();)
        ((Edge) enum.nextElement ()).delete (true);
    for (Enumeration enum = outEdges.elements (); enum.hasMoreElements
();)
        ((Edge) enum.nextElement ()).delete (false);
    inEdges.removeAllElements ();
    outEdges.removeAllElements ();
    removeFromParent ();
}
} // End of the class Vertex
```

287

```java
package caps.Parser;

import java.awt.Point;
import java.io.StringReader;
import java.io.StringWriter;
import caps.Psdl.*;
import java.util.Enumeration;
import java.util.Vector;

public class CreatePsdl {

static StringWriter writer;

public static void ReInit (StringWriter r)
{
    writer = new StringWriter ();
}

public static String getPsdl ()
{
    return writer.toString ();
}

public static void build (Vertex root, DataTypes types)
{
    DataFlowComponent dfc;
    dataTypes (types);
    for (Enumeration enum = root.breadthFirstEnumeration ();
enum.hasMoreElements ();) {
        dfc = (DataFlowComponent) enum.nextElement ();
        if (dfc instanceof Vertex && !(dfc instanceof External))
            operator ((Vertex) dfc);
    }
}

public static void dataTypes (DataTypes types)
{
    writer.write (types.toString ());
}

public static void operator (Vertex v)
{
    operatorSpecification (v);
    operatorImplementation (v);
}

public static void operatorSpecification (Vertex v)
{
    writer.write (v.getSpecification (true) + "\n\n");
}

public static void operatorImplementation (Vertex v)
{
    if (v.isLeaf ()) {
        writer.write ("          IMPLEMENTATION " + v.getImpLanguage () + " " +
v.getLabel () +          "_" + v.getId () + "\n");
        writer.write ("          END\n\n");
    }
    else {
        writer.write ("          IMPLEMENTATION\n");
        psdlImplementation (v);
        writer.write ("          END\n\n");
    }
}

public static void psdlImplementation (Vertex v)
{
    dataFlowDiagram (v);
    streams (v);
    timers (v);
    controlConstraints (v);
    informalDesc (v);
}

public static void dataFlowDiagram (Vertex v)
{
    writer.write ("          GRAPH\n");
    DataFlowComponent d;
    for (Enumeration enum = v.children (); enum.hasMoreElements ();) {
        d = (DataFlowComponent) enum.nextElement ();
        if (d instanceof Vertex && !(d instanceof External))
            vertex ((Vertex) d);
    }
    for (Enumeration enum = v.children (); enum.hasMoreElements ();) {
        d = (DataFlowComponent) enum.nextElement ();
        if (d instanceof Edge)
            edge ((Edge) d);
    }
}

public static void vertex (Vertex v)
{
    writer.write ("          VERTEX " + v.getLabel () + "_" + v.getId ()
+ "_" + (v.getId () - 1) + "  ");
    if (v.getMet () != null)
        writer.write (": " + v.getMet ().toString ());
    writer.write ("\n");
    vertexProperties (v);
}

public static void vertexProperties (Vertex v)
{
    writer.write ("          PROPERTY x = " + v.getX () + "\n");
```

```java
    writer.write ("
                            PROPERTY y = " + v.getY () + "\n");
    writer.write ("         PROPERTY radius = " + (v.getWidth () / 2)
+ "\n");
    writer.write ("         PROPERTY color = " + v.getColor () +
"\n");
    writer.write ("         PROPERTY label_font = " +
v.getLabelFontIndex () + "\n");
    writer.write ("         PROPERTY label_x_offset = " +
v.getLabelXOffset () + "\n");
    writer.write ("         PROPERTY label_y_offset = " +
v.getLabelYOffset () + "\n");
    writer.write ("         PROPERTY met_font = " + v.getMetFontIndex
() + "\n");
    writer.write ("         PROPERTY met_unit = " +
((v.getMet () == null) ? 1 : v.getMet ().getTimeUnits
()) + "\n");
    writer.write ("         PROPERTY met_x_offset = " +
v.getMetXOffset () + "\n");
    writer.write ("         PROPERTY met_y_offset = " +
v.getMetYOffset () + "\n");
    writer.write ("         PROPERTY is_terminator = " +
v.isTerminator () + "\n");
}

public static void edge (Edge e)
{
    writer.write ("         EDGE " + e.getLabel () + " ");
    if (e.getMet () != null)
        writer.write (": " + e.getMet ().toString () + " ");
    if (e.getSource () instanceof External)
        writer.write (e.getSource ().getLabel ());
    else
        writer.write (e.getSource ().getLabel () + "_" + e.getSource
().getId () + "_" + (e.getSource ().getId () - 1));
    writer.write (" -> ");
    if (e.getDestination () instanceof External)
        writer.write (e.getDestination ().getLabel ());
    else
        writer.write (e.getDestination ().getLabel () + "_" +
e.getDestination ().getId () + "_" +
(e.getDestination ().getId () - 1));
    writer.write ("\n");
    edgeProperties (e);
}

public static void edgeProperties (Edge e)
{
    Vector points = e.getPoints ();
    Point p = (Point) points.elementAt (points.size () / 2);;
    writer.write ("         PROPERTY id = " + e.getId () + "\n");
    writer.write ("         PROPERTY label_font = " +
e.getLabelFontIndex () + "\n");
    writer.write ("         PROPERTY label_x_offset = " +
e.getLabelXOffset () + "\n");
    writer.write ("         PROPERTY label_y_offset = " +
e.getLabelYOffset () + "\n");
    writer.write ("         PROPERTY latency_font = " +
e.getMetFontIndex () + "\n");
    writer.write ("         PROPERTY latency_unit = " +
((e.getMet () == null) ? 1 : e.getMet ().getTimeUnits
()) + "\n");
    writer.write ("         PROPERTY latency_x_offset = " +
e.getMetXOffset () + "\n");
    writer.write ("         PROPERTY latency_y_offset = " +
e.getMetYOffset () + "\n");
    writer.write ("         PROPERTY spline = \" ");
    if (e.getSource () instanceof External) {
        p = (Point) points.firstElement ();
        writer.write (p.x + " " + p.y + " ");
    }
    for (Enumeration enum = points.elements (); enum.hasMoreElements
();) {
        p = (Point) enum.nextElement ();
        if (!p.equals (points.firstElement ()) && !p.equals
(points.lastElement ())) { // do nothing
            if (!p.equals (points.firstElement ()))
                p = (Point) enum.nextElement ();
            if (!p.equals (points.lastElement ()))
                writer.write (p.x + " " + p.y + " ");
        }
    }
    if (e.getDestination () instanceof External) {
        p = (Point) points.lastElement ();
        writer.write (p.x + " " + p.y + " ");
    }
    writer.write ("\"\n");
}

public static void streams (Vertex v)
{
    DataFlowComponent d;
    String str = "";
    for (Enumeration enum = v.children (); enum.hasMoreElements ();) {
        d = (DataFlowComponent) enum.nextElement ();
        if (d instanceof Edge) {
            Edge e = (Edge) d;
            if (str.lastIndexOf ("_" + e.getLabel () + " ") == -1) {
                if (str.length () == 0) {
                    str = str.concat ("            " + e.getLabel () + " : "
+ e.getStreamType ());
                    if (!e.isStateStream ())
```

```java
            else if (!e.isStateStream ())
                str = str.concat (",\n                " + e.getLabel () + " : "
        + e.getStreamType ());
            }
        }
        if (str.length () != 0) {
            writer.write ("        DATA STREAM\n");
            writer.write (str + "\n");
        }
    }

    public static void timers (Vertex v)
    {
        if (v.getTimerList ().size () != 0)
            writer.write ("        TIMER " + v.extractList (v.getTimerList ())
    + "\n");
    }

    public static void controlConstraints (Vertex v)
    {
        writer.write ("        CONTROL CONSTRAINTS\n");
        DataFlowComponent d;
        for (Enumeration enum = v.children (); enum.hasMoreElements ();) {
            d = (DataFlowComponent) enum.nextElement ();
            if (d instanceof Vertex && !(d instanceof External))
                constraint ((Vertex) d);
        }
    }

    public static void constraint (Vertex v)
    {
        writer.write ("        OPERATOR " + v.getLabel () + "_" + v.getId
    () + "_" + (v.getId () - 1) + "\n");
        trigger (v);
        period (v);
        finishWithin (v);
        mcp (v);
        mrt (v);
        outputGuards (v);
        exceptionGuards (v);
        timerOps (v);
    }

    public static void trigger (Vertex v)
    {
        if (v.getTriggerType () != Vertex.UNPROTECTED) {
            writer.write ("            TRIGGERED BY ");
        if (v.getTriggerType () == Vertex.BY_SOME)
            writer.write ("SOME ");
        else
            writer.write ("ALL ");
```

```java
            writer.write (v.extractList (v.getTriggerStreamsList ()) + "\n");
        if (v.getTriggerReqmts ().size () != 0)
            writer.write ("                    REQUIRED BY " + v.extractList
    (v.getTriggerReqmts ()));
        }
    }

    public static void period (Vertex v)
    {
        if (v.getPeriod () != null) {
            writer.write ("        PERIOD " + v.getPeriod ().toString ()
    + "\n");
        if (v.getPeriodReqmts ().size () != 0)
            writer.write ("                    REQUIRED BY " + v.extractList
    (v.getPeriodReqmts ()) + "\n");
        }
    }

    public static void finishWithin (Vertex v)
    {
        if (v.getFinishWithin () != null) {
            writer.write ("        FINISH WITHIN " + v.getFinishWithin
    ().toString () + "\n");
        if (v.getFinishWithinReqmts ().size () != 0)
            writer.write ("                    REQUIRED BY " + v.extractList
    (v.getFinishWithinReqmts ()) + "\n");
        }
    }

    public static void mcp (Vertex v)
    {
        if (v.getMcp () != null) {
            writer.write ("        MINIMUM CALLING PERIOD " + v.getMcp
    ().toString () + "\n");
        if (v.getMcpReqmts ().size () != 0)
            writer.write ("                    REQUIRED BY " + v.extractList
    (v.getMcpReqmts ()) + "\n");
        }
    }

    public static void mrt (Vertex v)
    {
        if (v.getMrt () != null) {
            writer.write ("        MAXIMUM RESPONSE TIME " + v.getMrt
    ().toString () + "\n");
        if (v.getMrtReqmts ().size () != 0)
            writer.write ("                    REQUIRED BY " + v.extractList
    (v.getMrtReqmts ()) + "\n");
        }
    }

    public static void outputGuards (Vertex v)
```

```java
package caps.Parser;

import java.io.*;

public class GrammarCheck {

    public static StringReader reader;

    public static final int ID = 1;

    public static final int TYPE_NAME = 2;

    public static final int INTEGER_LITERAL = 3;

    public static final int INITIAL_EXPRESSION = 4;

    public static final int EXPRESSION = 5;

    public static final int CHECK_OUTPUT_GUARDS = 6;

    public static final int CHECK_EXCEPTION_GUARDS = 7;

    public static final int CHECK_EXCEPTION_LIST = 8;

    public static final int CHECK_TIMER_OPS = 9;

    public static final int INFORMAL_DESCRIPTION = 10;

    public static final int FORMAL_DESCRIPTION = 11;

    public static final int DATA_TYPE = 12;

    public static final int CHECK_PARENT_SPEC = 13;

    public static boolean isValid (String str, int kind)
    {
        reader = new StringReader (str);
        PsdlParser.ReInit (reader);
        boolean flag = true;
        try {
            switch (kind) {
                case ID :
                    PsdlParser.id ();
                    break;
                case TYPE_NAME :
                    PsdlParser.type_name ();
                    break;
                case INTEGER_LITERAL :
                    PsdlParser.integer_literal ();
                    break;
                case INITIAL_EXPRESSION :
                    PsdlParser.initial_expression ();
```

```java
{
    if (v.getOutputGuardList ().length () != 0) {
        String str = v.getOutputGuardList ();
        writer.write (v.extractString (str, true));
    }
}

public static void exceptionGuards (Vertex v)
{
    if (v.getExceptionGuardList ().length () != 0) {
        String str = v.getExceptionGuardList ();
        writer.write (v.extractString (str, true));
    }
}

public static void timerOps (Vertex v)
{
    if (v.getTimerOpList ().length () != 0) {
        String str = v.getTimerOpList ();
        writer.write (v.extractString (str, true));
    }
}

public static void informalDesc (Vertex v)
{
    if (v.getGraphDesc ().length () != 0) {
        String str = v.getGraphDesc ();
        writer.write (v.extractString (str, false));
    }
}

} // End of the class CreatePsdl
```

```java
            break;
        case EXPRESSION :
            PsdlParser.expression ();
            break;
        case CHECK_OUTPUT_GUARDS :
            PsdlParser.check_output_guards ();
            break;
        case CHECK_EXCEPTION_GUARDS :
            PsdlParser.check_exception_guards ();
            break;
        case CHECK_EXCEPTION_LIST :
            PsdlParser.check_exception_list ();
            break;
        case CHECK_TIMER_OPS :
            PsdlParser.check_timer_ops ();
            break;
        case INFORMAL_DESCRIPTION :
            PsdlParser.informal_desc ();
            break;
        case FORMAL_DESCRIPTION :
            PsdlParser.formal_desc ();
            break;
        case DATA_TYPE :
            PsdlParser.psdl ();
            break;
        case CHECK_PARENT_SPEC :
            PsdlParser.check_parent_spec ();
            break;
        default :
            break;
        }
        if (PsdlParser.getNextToken ().kind != PsdlParserConstants.EOF) {
// If there is not only one id
            System.out.println ("Characters encountered after a valid
token");
            flag = false;
        }
    } catch (ParseException e) {
        System.out.println ("Parse exception occurred");
        System.out.println (e);
        flag = false;
    } catch (TokenMgrError e) {
        System.out.println ("Lexical error occurred");
        System.out.println (e);
        flag = false;
    } catch (Exception e) {
        System.out.println ("An error occurred during parsing the
structure");
        System.out.println (e);
        flag = false;
    }

    return flag;
}
```

LIST OF REFERENCES

[1]   Luqi and R.Steigerwald. Rapid Software Prototyping. IEEE Software, 1992.

[2]   Luqi, Valdins Berzins and Raymond T. Yeah. A Prototyping Language for Real-Time Software. IEEE Transactions on Software Engineering. October 1988.

[3]   Luqi and Mohammed Ketabchi. A Computer Aided Prototyping System. IEE Software, March 1988.

[4]   Kenneth B. Moeller. Evolution of a Graphical User Interface for the Rapid Prototyping of Real-Time Embedded Systems. Master's thesis. Naval Postgraduate School. September 1997.

[5]   The Java (TM) Development Kit 1.2, http://www.javasoft.com/products/jdk/1.2/ , March 1999

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
    8725 John J. Kingman Road, Ste 0944
    Ft. Belvoir, VA 22060-6218

2. Deniz Kuvvetleri Komutanligi 2
    Personel Daire Baskanligi
    Bakanliklar
    Ankara, TURKEY

3. Deniz Harp Okulu Komutanligi 1
    Kutuphane
    Tuzla
    Istanbul, TURKEY

4. Dudley Knox Library 2
    Naval Postgraduate School
    411 Dyer Rd.
    Monterey, CA 93943-5101

5. Chairman, Department of Computer Science 1
    Code CS
    Naval Postgraduate School
    Monterey, CA 93943

6. Dr. Mantak Shing, Code CS/Sh 1
    Computer Science Department
    Naval Postgraduate School
    Monterey, CA 93943

7. Dr. Valdis Berzins, Code CS/Be 1
    Computer Science Department
    Naval Postgraduate School
    Monterey, CA 93943

8. Prof. Luqi, Code CS/Lq 1
    Computer Science Department
    Naval Postgraduate School
    Monterey, CA 93943

9. LTJG Ilker Duranlioglu 3
    68 Sokak No 13/12
    Uckuyular
    Izmir, TURKEY