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.
IMPLEMENTATION OF A PORTABLE PSDL EDITOR FOR THE HETEROGENEOUS SYSTEMS INTEGRATOR

Duranlioglu, Ilker

Naval Postgraduate School
Monterey, CA 93943-5000

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.

Approved for public release; distribution is unlimited.

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.
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
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

A. COMPUTER AIDED PROTOTYPING SYSTEM .................................................. 2
B. CAPS RELEASE 2.0 GRAPH EDITOR ......................................................... 4
C. RESEARCH GOAL ....................................................................................... 7

## II. REDESIGNING CAPS MAIN PROGRAM ......................................................... 9

A. PROTOTYPE DIRECTORY STRUCTURE ......................................................... 9
B. CREATING A NEW PROTOTYPE .................................................................. 9
C. OPENING AN EXISTING PROTOTYPE ...................................................... 10
D. INVOKING THE GRAPH EDITOR .............................................................. 10
E. EXITING THE PROGRAM .......................................................................... 10

## III. USER INTERFACE DESIGN .................................................................. 11

A. ARCHITECTURE OVERVIEW .................................................................... 11
B. DATA STRUCTURES .................................................................................. 14
C. PSDL MAPPING ....................................................................................... 17
D. SYNTACTIC VALIDATIONS ...................................................................... 17

## IV. USER INTERFACE IMPLEMENTATION .................................................... 19

A. PSDL EDITOR ENVIRONMENT .................................................................. 19
B. DATA FLOW DIAGRAM .......................................................................... 28
C. CHANGING COMPONENTPERTIES ......................................................... 42

## V. CONCLUSION ......................................................................................... 45

APPENDIX A PSDL GRAMMAR ...................................................................... 49
APPENDIX B PSDLBUILDER.JJ AND PSDLGRAMMAR.JJ ......................... 57
APPENDIX C DOCUMENTATION .................................................................. 83
APPENDIX D SOURCE CODE ...................................................................... 201

LIST OF REFERENCES .................................................................................. 293

INITIAL DISTRIBUTION LIST ...................................................................... 295
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.
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
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
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
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
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.
Figure 7. Data Flow Between Major Modules
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 implemented 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
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.
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.
<table>
<thead>
<tr>
<th>Menu</th>
<th>Identification</th>
<th>Functionality</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Save</td>
<td>Saves the current prototype to the disk.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Restore From Save</td>
<td>Ignores changes and restores the data flow diagram from the prototype file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Print</td>
<td>Prints the data flow diagram.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exit</td>
<td>Quits the PSDL Editor. Prompts for save.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Edit</td>
<td>Undo for the last action.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Redo for the last action.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Selects the selected component.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Deletes the selected component.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Changes the current color of the draw panel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Changes the current font of the draw panel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Hides or views the tree panel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Menu Items
<table>
<thead>
<tr>
<th>Menu</th>
<th>Identification</th>
<th>Functionality</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>Tool Tips</td>
<td>Enables or disables the tool tips for the toolbar buttons.</td>
<td>Always</td>
</tr>
<tr>
<td></td>
<td>Auto Select Mode</td>
<td>If not checked, component insertion to the draw panel will be continuous.</td>
<td>Always</td>
</tr>
<tr>
<td></td>
<td>Refresh</td>
<td>Refreshes the components on the draw panel.</td>
<td>Always</td>
</tr>
<tr>
<td></td>
<td>Go to Root</td>
<td>Changes the level to the root operator.</td>
<td>Not available if the current operator is the root operator.</td>
</tr>
<tr>
<td></td>
<td>Go to Parent</td>
<td>Changes the level to the parent of the current operator.</td>
<td>Not available if the current operator is the root operator.</td>
</tr>
<tr>
<td></td>
<td>Decompose</td>
<td>Decomposes the current operator.</td>
<td>Not available if the selected component is a stream</td>
</tr>
<tr>
<td></td>
<td>PSDL</td>
<td>Provides help for PSDL Grammar</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Help</td>
<td>Provides help for Operators</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Streams</td>
<td>Provides help for Streams</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exceptions</td>
<td>Provides help for Exceptions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Timers</td>
<td>Provides help for timers</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Menu Items
<table>
<thead>
<tr>
<th>Button</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
<td>Draws an operator into the drawing panel</td>
</tr>
<tr>
<td>TERM</td>
<td>Draws a terminator into the drawing panel</td>
</tr>
<tr>
<td></td>
<td>Draws a stream into the drawing panel</td>
</tr>
<tr>
<td></td>
<td>Selects a component from the drawing panel</td>
</tr>
<tr>
<td>TYPE</td>
<td>Opens text editor to view or edit Data Types</td>
</tr>
<tr>
<td>SPEC</td>
<td>Opens text editor to view or edit Parent Specs</td>
</tr>
<tr>
<td></td>
<td>Opens id-list editor to view or edit Timers</td>
</tr>
<tr>
<td>DESC</td>
<td>Opens text editor to view or edit Informal Graph Description</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Component</th>
<th>Icon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composite operator</td>
<td>○</td>
</tr>
<tr>
<td>Atomic operator</td>
<td>○</td>
</tr>
<tr>
<td>Composite terminator</td>
<td>□</td>
</tr>
<tr>
<td>Atomic terminator</td>
<td>□</td>
</tr>
<tr>
<td>Stream</td>
<td>↑</td>
</tr>
<tr>
<td>State stream</td>
<td>↑</td>
</tr>
</tbody>
</table>

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.

\textit{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.
<table>
<thead>
<tr>
<th>Menu</th>
<th>Menu Item</th>
<th>Hot Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Save</td>
<td>CTRL-S</td>
</tr>
<tr>
<td></td>
<td>Print</td>
<td>CTRL-P</td>
</tr>
<tr>
<td>Edit</td>
<td>Undo</td>
<td>CTRL-Z</td>
</tr>
<tr>
<td></td>
<td>Redo</td>
<td>CTRL-Y</td>
</tr>
<tr>
<td></td>
<td>Select All</td>
<td>CTRL-A</td>
</tr>
<tr>
<td></td>
<td>Delete</td>
<td>DEL</td>
</tr>
<tr>
<td>View</td>
<td>Refresh</td>
<td>CTRL-F</td>
</tr>
<tr>
<td>PSDL</td>
<td>Go to Root</td>
<td>CTRL-R</td>
</tr>
<tr>
<td></td>
<td>Go to Parent</td>
<td>CTRL-O</td>
</tr>
<tr>
<td></td>
<td>Decompose</td>
<td>CTRL-D</td>
</tr>
</tbody>
</table>

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.
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**
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.

\textit{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.

\textit{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
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
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
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
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.
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
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.

\textbf{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.

\section*{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
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.
Figure 19. Operator Properties Dialog

Figure 20. Stream Properties Dialog

Figure 21. Tool Bar Buttons
<table>
<thead>
<tr>
<th>Window</th>
<th>Index</th>
<th>Component name</th>
<th>Component</th>
<th>Component functionality</th>
<th>Validation rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator</td>
<td>1</td>
<td>Operator name</td>
<td>Data entry</td>
<td>Operator name is entered into the text area</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Operator type</td>
<td>Combo box</td>
<td>Changes an operator to terminator or vice versa</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Implementation language</td>
<td>Combo box</td>
<td>Changes implementation language to Ada or Tae</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Trigger type</td>
<td>Combo box</td>
<td>Unprotected, by some or by all</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Trigger if condition</td>
<td>Text editor</td>
<td>Opens text editor to edit trigger if condition</td>
<td>expression</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Trigger required by</td>
<td>Id-list editor</td>
<td>Opens id-list editor to edit trigger requirements</td>
<td>id_list</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Trigger identifier list</td>
<td>Id-list editor</td>
<td>Opens id-list editor to edit trigger streams list</td>
<td>id_list</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Trigger if condition exp.</td>
<td>Display only</td>
<td>If condition exp. is displayed in the text area</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>Timing type</td>
<td>Combo box</td>
<td>Not-time-critical, periodic or sporadic</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>MET Value</td>
<td>Data entry</td>
<td>MET value is entered into the text area</td>
<td>integer_literal</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>Period/MCP Value</td>
<td>Data entry</td>
<td>Period or MCP value is entered into the text area</td>
<td>integer_literal</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>Finish within/MRT value</td>
<td>Data entry</td>
<td>Finish within/MRT value is entered here</td>
<td>integer_literal</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>MET units</td>
<td>Combo box</td>
<td>Microsec, ms, sec, min or hours</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>Period/MCP units</td>
<td>Combo box</td>
<td>Microsec, ms, sec, min or hours</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>Finish within/MRT units</td>
<td>Combo box</td>
<td>Microsec, ms, sec, min or hours</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>MET required by</td>
<td>Id-list editor</td>
<td>Opens id-list editor to edit met requirements</td>
<td>id_list</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>Period/MCP required by</td>
<td>Id-list editor</td>
<td>Opens id-list editor to edit period requirements</td>
<td>id_list</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>Finish within/MRT req.by</td>
<td>Id-list editor</td>
<td>Opens id-list editor to edit FW-MRT requirement</td>
<td>id_list</td>
</tr>
</tbody>
</table>

Table 5. Component Properties Dialog Functionality
<table>
<thead>
<tr>
<th>Window</th>
<th>Index</th>
<th>Component name</th>
<th>Component</th>
<th>Component functionality</th>
<th>Validation rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operator</td>
<td>19</td>
<td>Output guards</td>
<td>Text editor</td>
<td>Opens text editor to edit operator output guards</td>
<td>check_output Guards</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>Exception list</td>
<td>Text editor</td>
<td>Opens text editor to edit operator exceptions</td>
<td>check_exception_list</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>Exception guards</td>
<td>Text editor</td>
<td>Opens text editor to edit operator output guards</td>
<td>check_exception Guards</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>Timer ops</td>
<td>Id-list editor</td>
<td>Opens id-list editor to edit operator timer-ops</td>
<td>check_timer_ops</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>Keywords</td>
<td>Id-list editor</td>
<td>Opens id-list editor to edit operator keywords</td>
<td>id_list</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>Informal description</td>
<td>Text editor</td>
<td>Opens text editor to edit operator informal desc.</td>
<td>check_informal_desc</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>Formal Description</td>
<td>Text editor</td>
<td>Opens text editor to edit operator formal desc.</td>
<td>check_informal_desc</td>
</tr>
<tr>
<td>Stream</td>
<td>26</td>
<td>Stream name</td>
<td>Data entry</td>
<td>Stream name is entered into the text area</td>
<td>id_list</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>Stream type</td>
<td>Data entry</td>
<td>Stream type is entered into the text area</td>
<td>type_name</td>
</tr>
<tr>
<td></td>
<td>28</td>
<td>State stream selection</td>
<td>Radio button</td>
<td>Sets the stream type to state stream or not</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>State stream initial value</td>
<td>Text editor</td>
<td>Opens text editor to edit state stream initial value</td>
<td>initial_expression</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>State initial value display</td>
<td>Display only</td>
<td>Shows the state initial value</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>Latency Value</td>
<td>Data entry</td>
<td>Latency value is entered into the text area</td>
<td>integer literal</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>Latency units</td>
<td>Combo box</td>
<td>Microsec, ms, sec, min or hours</td>
<td>n/a</td>
</tr>
<tr>
<td>Tool Bar</td>
<td>33</td>
<td>Data Types</td>
<td>Text editor</td>
<td>Opens text editor to edit data types</td>
<td>psdl</td>
</tr>
<tr>
<td></td>
<td>34</td>
<td>Parent Specifications</td>
<td>Text editor</td>
<td>Opens text editor to edit parent specifications</td>
<td>check_parent_spec</td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>Timers</td>
<td>Id-list editor</td>
<td>Opens id-list editor to edit timers</td>
<td>id_list</td>
</tr>
<tr>
<td></td>
<td>36</td>
<td>Graph Description</td>
<td>Text editor</td>
<td>Opens text editor to edit graph description</td>
<td>check_informal_desc</td>
</tr>
</tbody>
</table>

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 than 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.
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
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.

* This appendix is taken from Reference 4, Appendix A.
1. \[ <\text{psdl}> \]
   \[ ::= \{ <\text{component}> \} \]

2. \[ <\text{component}> \]
   \[ ::= <\text{data_type}> \]
   \[ | <\text{operator}> \]

3. \[ <\text{data_type}> \]
   \[ ::= \text{type} <\text{id}> <\text{type_spec}> <\text{type_impl}> \]

4. \[ <\text{type_spec}> \]
   \[ ::= \text{specification} [ \text{generic} <\text{type_decl}> ] [ <\text{type_decl}> \]
   \[ \{ <\text{operator}> <\text{op_name}> <\text{operator_spec}> \} \]
   \[ [ <\text{functionality}> ] \text{end} \]

5. \[ <\text{operator}> \]
   \[ ::= \text{operator} <\text{op_name}> <\text{operator_spec}> <\text{operator_impl}> \]

6. \[ <\text{operator_spec}> \]
   \[ ::= \text{specification} \{ <\text{interface}> \} [ <\text{functionality}> ] \text{end} \]

7. \[ <\text{interface}> \]
   \[ ::= <\text{attribute}> [ <\text{reqmts_trace}> ] \]

8. \[ <\text{attribute}> \]
   \[ ::= \text{generic} <\text{type_decl}> \]
   \[ | \text{input} <\text{type_decl}> \]
   \[ | \text{output} <\text{type_decl}> \]
   \[ | \text{states} <\text{type_decl}> \text{initially} <\text{initial_expression_list}> \]
exceptions <id_list>
maximum execution time <time>

9. \(<\text{type_decl}\) :
\[\text{\textbar}\]
\[\text{:=} \ <\text{id}\> : <\text{name}\> \{, <\text{id}\> : <\text{name}\> \}

10. \(<\text{name}\) :
\[\text{:=} \ <\text{id}\>
\[\text{\textbar}\]
\[<\text{id}\> \[\text{\textbar}\] \text{\textquotesingle}[ <\text{type_decl}\> \text{\textquotesingle]}

11. \(<\text{id_list}\) :
\[\text{:=} \ <\text{id}\> \{, <\text{id}\> \}

12. \(<\text{reqmts_trace}\) :
\[\text{:=} \ \text{required by} <\text{id_list}\>

13. \(<\text{functionality}\) :
\[\text{:=} \ [ <\text{keywords}\> ] [ <\text{informal_desc}\> ] [ <\text{formal_desc}\> ]

14. \(<\text{keywords}\) :
\[\text{:=} \ \text{keywords} <\text{id_list}\>

15. \(<\text{informal_desc}\) :
\[\text{:=} \ \text{description} \{ \text{\textquotesingle} <\text{text}\> \text{\textquotesingle} \}

16. \(<\text{formal_desc}\) :
\[\text{:=} \ \text{axioms} \{ <\text{text}\> \text{\textquotesingle} \}

17. \(<\text{type_impl}\) :
\[\text{:=} \ \text{implementation} <\text{id}\> <\text{id}\> \text{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.  \[ \langle \text{expression\_list} \rangle \]

\[ ::= \langle \text{expression} \rangle \{ , \langle \text{expression} \rangle \} \]

39.  \[ \langle \text{expression\_list} \rangle \]

\[ ::= \text{true} \]
\[ \mid \text{false} \]
\[ \mid \langle \text{integer\_literal} \rangle \]
\[ \mid \langle \text{time} \rangle \]
\[ \mid \langle \text{real\_literal} \rangle \]
\[ \mid \langle \text{string\_literal} \rangle \]
\[ \mid \langle \text{id} \rangle \]
\[ \mid \langle \text{type\_name} \rangle \cdot \langle \text{op\_name} \rangle \ [ \langle \text{expression\_list} \rangle
\[ \cdot \rangle ] \]
\[ \mid \langle \text{expression} \rangle \langle \text{binary\_op} \rangle \langle \text{expression} \rangle \]
\[ \mid \langle \text{unary\_op} \rangle \langle \text{expression} \rangle \]

40.  \[ \langle \text{op\_name} \rangle \]

\[ ::= \langle \text{id} \rangle \]

41.  \[ \langle \text{id} \rangle \]

\[ ::= \langle \text{letter} \rangle \{ \langle \text{alpha\_numeric} \rangle \} \]

42.  \[ \langle \text{real\_literal} \rangle \]

\[ ::= \langle \text{integer\_literal} \rangle \cdot \langle \text{integer\_literal} \rangle \]

43.  \[ \langle \text{integer\_literal} \rangle \]

\[ ::= \langle \text{digit} \rangle \{ \langle \text{digit} \rangle \} \]
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. PSDLParse and PSDLBuilder

PSDLParse 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.

PSDLParse 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 PSDLParse, 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.
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:
{
    |
    | "\" |
    | "\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" >
    | < MICROSET : "microset" >
    | < 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_LITERAL : ["\"", "]" ] >
  | < 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 pddl () :
() ()
| 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 () [ regmats_trace () ]
}

/**
 * Production 8
 */
void attribute () {
    <GENERIC> type_decl ()
    | <INPUT> type_decl ()
    | <OUTPUT> type_decl ()
    | <STATES> type_decl () <INITIAL> 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 modified to remove common prefix id () */
void type_name () {
    id ()
    | id () [" type_decl () "]
}

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

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

/**
 * Production 12
 */
void regmats_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 () {
}
** 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 () )

/* Production 18
/* 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 19
 */
void psdl_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 a left recursion problem and is modified */
/**
 * void initial_expression () : () {
  < TRUE >
  | < FALSE >
  | < INTEGERLiteral >
  | < 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 >
  | < STRINGLiteral >
  < 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
```c
/* Production 35 */
void unary_op () :
{
    <NOT> | <ABS> | <MINUS> | <PLUS>
}

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

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

/* Production 38 */
void expression_list () :
{
    expression () {[ "."] expression ()}
}

/* Production 39 */
/* This production has two common prefix problems and a left recursion problem and is modified */

void expression () :
{
    <TRUE>
    | <FALSE>
    | time ()
    | <INTEGER_LITERAL>
    | <REAL_LITERAL>
    | <STRING_LITERAL>
    id ()
    type_name () [ "."] cp_name () [ "."] expression_list ()]
    [ "."] expression ()
    expression () binary_op () expression ()
    unary_op () expression ()
}

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

void expression_l () :
{
    < 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 () :
```
void expression_suffix2 () :
{}  
empty_string();
    
    
| +. "< INTEGER_LITERAL >
| unit |


*/
/* 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_l literal () :
{}  
| < INTEGER_LITERAL > +. "< INTEGER_LITERAL >

*/
/* Production 43 */
/* This is a token, it is removed from the parser for efficiency
void integer_l literal () :
{}  
| < INTEGER_LITERAL >

*/
/* Production 44 */
/* This is a token, it is removed from the parser for efficiency
void string_l 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 PDDL 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 () [regmts_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 () ] [regmts_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 () ] [regmts_trace ()] )
  )
/**
* This production is not in the grammar
* It is used to check parent specs
*/
void check_parent_spec () {
  ( < OPERATOR > op_name () operator_spec ()
  )

66
/*
 * Program : PadlBuilder.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 (PadlBuilder)

package caps.Builder;

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

public class PadlBuilder {

  private static Vector dfoVector;
  private static Vector streamVector;
  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);
    }
    dfoVector = new Vector ();
    streamVector = new Vector ();
    idList = new Vector ();
    currentOp = null;
    currentEdge = null;
    currentChild = null;
    ReInit (reader);
    try {
      push ();
    } catch (ParseException ex) {
      System.out.println (ex);
      System.exit (0);
    }
    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 = dfoVector.elements (); enum.hasMoreElements ();) {
      d = (DataFlowComponent) enum.nextElement ();
      String sstr = "";
      if (doubleSuffix)
        sstr = new String (d.getLabel () + "\" + d.getId () + "\" + (d.getId () - 1));
      else
        sstr = new String (d.getLabel () + "\" + d.getId ());
      if ((id 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
      }
    }
    return found;
  }

public static String label;
public static int id;

public static Vertex buildPrototype (File file) {
  BufferedReader reader = null;
  try {
    reader = new BufferedReader (new FileReader (file));
  } catch (FileNotFoundException e) {
    System.out.println (e);
  }
  dfoVector = new Vector ();
  streamVector = new Vector ();
  idList = new Vector ();
  currentOp = null;
  currentEdge = null;
  currentChild = null;
  ReInit (reader);
  try {
    push ();
  } catch (ParseException ex) {
    System.out.println (ex);
    System.exit (0);
  }
  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 = dfoVector.elements (); enum.hasMoreElements ();) {
    d = (DataFlowComponent) enum.nextElement ();
    String sstr = "";
    if (doubleSuffix)
      sstr = new String (d.getLabel () + "\" + d.getId () + "\" + (d.getId () - 1));
    else
      sstr = new String (d.getLabel () + "\" + d.getId ());
    if ((id 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
    }
  }
  return found;
}

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 = dfoVector.elements (); enum.hasMoreElements ();) {
    d = (DataFlowComponent) enum.nextElement ();
    String sstr = "";
    if (doubleSuffix)
      sstr = new String (d.getLabel () + "\" + d.getId () + "\" + (d.getId () - 1));
    else
      sstr = new String (d.getLabel () + "\" + d.getId ());
    if ((id 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
    }
  }
  return found;
}
found.setLabel (label);
found.setId (id);
dfvVector.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 ());
um = 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.getLabel () == null)
    return found;
}
return found;
}

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

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 ());
}
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 () == null)
    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.set (Integer.parseInt (exp));
else if (prop.equalsIgnoreCase ('y'))
  v.setY (Integer.parseInt (exp));
else if (prop.equalsIgnoreCase ('radius'))
  v.setRadius (Integer.parseInt (exp) * 2);
}
else if (prop.equalsIgnoreCase("color"))
    v.setColor (Integer.parseInt (exp));
else if (prop.equalsIgnoreCase("label_font"))
    v.setFontIndex (Integer.parseInt (exp));
else if (prop.equalsIgnoreCase("label_x_offset"))
    v.setLabelIndex (Integer.parseInt (exp));
else if (prop.equalsIgnoreCase("label_y_offset"))
    v.setLabelIndex (Integer.parseInt (exp));
else if (prop.equalsIgnoreCase("met_font"))
    v.setFontIndex (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.setFontIndex (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase("label_x_offset"))
        e.setLabelXIndex (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase("label_y_offset"))
        e.setLabelYIndex (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase("latency_font"))
        e.setFontIndex (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.setXNodeOffset (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase("latency_y_offset"))
        e.setYNodeOffset (Integer.parseInt (exp));
    else if (prop.equalsIgnoreCase("spline"))
        e.setInitialControlPoints (exp);
}

} // End of the class PeddBuilder

PARSER_END (peddBuilder):

/* 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" >
    | < EXCEPTION : "exceptions" >
    | < DESCRIPTION : "description" >
    | < SPECIFICATION : "specification" >
    | < IMPLEMENTATION : "implementation" >
/* Operators */
TOKEN:
{
  /* Binary Operators */
  < OR : "or" >
  < AND : "and" >
  < NOD : "not" >
  < REM : "rem" >
  < XOR : "xor" >
  < GREATER_THAN : ">" >
  < LESS_THAN : "<" >
  < EQUALS : "=" >
  < GREATER_EQUAL_TO : "\>\=" >
  < LESS_EQUAL_TO : "\<\=" >
  < DIVIDE_EQUALS : "/\=" >
  < FLOWS : ">>" >
  < MINUS : "\-\-" >
  < AMPERCENT : "\&\&" >
  < STAR : "\*\*" >
  < FACTOR : "/\/" >
  /* Unary Operators */
  < ABS : "abs" >
  < NOT : "not" >
}

/* String, real, and integer literals */
TOKEN:
{
  < TEXT : "" { < CHAR_TEXT > } "" >
  < #CHAR_TEXT : "\"" >
  < STRING_LITERAL : """" { < CHAR_LITERAL > } """" >
  < #CHAR_LITERAL : "\"\", "\"\" >
  < 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) ]

70


```java
/**
 * 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 () /* regmts_trace is under attribute */
}
/**
 * Production 8
 */
void attribute () :
{
    String initial;
    Token tok;
    Vector list;
    PSEUDO list;
    Vector regmts = null;
    String str;
    
    tok = <GENERIC> type_decl [false]  
    [ str = currentOp.getGenericList ();
    if (str != "")
        str = str.concat ('"' + str);  
        str = str.concat (tok.toString () + ' ' + extractIdList (idList));
    [ list = regmts_trace () ( str = str.concat ('"\n REQUIRED BY ' + extractIdList (list)); ) ]
    [ currentOp.setGenericList (str); ]
    | <INPUT> type_decl [false]  [ regmts = regmts_trace (); ]
    [ (Vector currentOp.getSpecRegs ()).elementAt (0)).addElement (extractIdList (regmts)); ]
    | <OUTPUT> type_decl [false]  [ regmts = regmts_trace (); ]
    [ (Vector currentOp.getSpecRegs ()).elementAt (2)).addElement (extractIdList (regmts)); ]
    | <STATE> type_decl [true]  <INITIAL> initial = initial_expression_list ()
    [ currentEdge.setStateStream (true);  
        currentEdge.setInitialValue (initial); ]
    | regmts = regmts_trace ()
    [ ((Vector currentOp.getSpecRegs ()).elementAt (2)).addElement (extractIdList (regmts)); ]
    | tok = <EXCEPTIONS> list = id_list ()
    [ str = currentOp.getExceptionList ();
    if (str != "")
        str = str.concat ('"' + str);  
        str = str.concat (tok.toString () + ' ' + extractIdList (list));
    [ list = regmts_trace () ( str = str.concat ('"\n REQUIRED BY ' + extractIdList (list)); ) ]
    [ currentOp.setExceptionList (str); ]
    | "maximum execution time" met = time () ( currentOp.setMet (met); )
    [ list = regmts_trace () ( currentOp.setSpecRegs (list); ) ]
}
/**
 * Production 9
 */
void type_decl (boolean buildEdge) :
{
    Vector idList;

```
String type = "";

if (idList == null) {
    (/* 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 modified 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 regs_trace () :
{ Vector list;

    "required by" list = id_list ()
        ( return list; )
}
/**
 * Production 13
 */
/*
 * This production is included directly in other productions,
 * because it causes empty string
 * void functionality () :
{ [] keywords () [informal_doc []] [ formal_doc []] */

72
*/

/** 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.setImplLanguage (language); )
    psdl_impl (); <END
}
/**
 * Production 19
 */
void pssl_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;
  PSSLTime met;
  [VERTEX] name = op_Id ()
  { currentChild = findOperator (name, true); }
  [ ':', met = time () { currentChild.setMet (met); } ] ( property (currentChild) )*
  [ currentChild = null; ]
 }/**
 * Production 22
 */
void edge ()
{
  String name;
  PSSLTime latency;
  Vertex src;
  Vertex dest;
  [EDGE] name = id ()
  { if ((currentEdge = findEdge (name)) == null) {
    currentEdge = new Edge (0, 0, currentOp);
    currentEdge.setLabel (name);
    streamVector.addElement (currentEdge);
  } else {
    if (currentEdge.getSource () == null) {
      streamVector.removeElement (currentEdge);
      currentEdge.removeFromParent ();
    }
    Edge e = new Edge (0, 0, currentOp);
    e.setLabel (name);
    e.setStreamType (currentEdge.getStreamType ());
    e.setStateStream (currentEdge.getStateStream ());
    e.setInitialValue (currentEdge.getInitialValue ());
    streamVector.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) )*)
void property (DataFlowComponent dfc) {
    String prop;
    String exp;
    
    <PROPERTY> prop = id () "*" exp = expression ()
    if (id() instanceof Vertex)
        setVertexProperty ((Vertex) dfc, prop, exp);
    else
        setEdgeProperty ((Edge) dfc, prop, exp);
}

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

```java
String initial_expression_1 () :
{
    Token tok;
    String str;
    String suffix = "";

    tok = < TRUE > ( return tok.toString (); )
    tok = < FALSE > ( return tok.toString (); )
    tok = < INTEGER_LITERAL > ( return tok.toString (); )
    tok = < STRING_LITERAL > suffix = initial_expression_suffix1 ()
        ( return new String (tok.toString () + suffix); )
    str = id () suffix = initial_expression_suffix2 ()
        ( return new String (str + suffix); )
    | token 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 ""; )
    | token tok = < INTEGER_LITERAL >
        ( return new String (".*" + tok.toString (); )
}

String initial-expression_suffix2 () :
{
    String str = "";
    String s = "";

    empty_string ()
        ( return ""; )
    | token s = type_name_suffix () "*
        ( str = new String ("*" + str + '*'); )
    | token s = op_name ()
        ( str = str.concat ("*" + op_name );
    | token 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 = < GREATER_THAN > ( return tok.toString (); )
    tok = < LESS_THAN > ( return tok.toString (); )
    tok = < EQV > ( return tok.toString (); )
    tok = < GTEQ > ( return tok.toString (); )
    tok = < LTEQ > ( return tok.toString (); )
    tok = < PLUS > ( return tok.toString (); )
    tok = < MINEQ > ( return tok.toString (); )
    tok = < PERCENT > ( return tok.toString (); )
    tok = < STAR > ( return tok.toString (); )
    tok = < FACTOR > ( return tok.toString (); )
    tok = < MOD > ( return tok.toString (); )
    tok = < RMD > ( 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; 
 t = < 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 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_l () expTail = expression_tail ()
 ( exp = exp.concat (expTail); return exp; )
}

String expression_l () :
{
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 ""; }
  
  if tok < INTEGER_LITERAL >
      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 () :
  {  
    

79
< 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 () [remsg_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 []] [remsg_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 []] [remsg_trace []]> 
}
APPENDIX C. DOCUMENTATION OF THE SOURCE CODE

caps.Caps, 87
caps.EditorDriver, 88
caps.CAPSMain.CAPSMainMenuBar, 89
caps.CAPSMain.CAPSMainWindow, 90
caps.CAPSMain.DataBasesMenu, 93
caps.CAPSMain.EditMenu, 94
caps.CAPSMain.ExecSupportMenu, 97
caps.CAPSMain.ExitCAPSMain, 98
caps.CAPSMain.HelpMenu, 100
caps.CAPSMain.PrototypeMenu, 101
caps.Display.DisplayComponent, 103
caps.Display.DisplayExternal, 107
caps.Display.DisplayVertex, 109
caps.Display.EdgePath, 112
caps.GraphEditor.ColorConstants, 115
caps.GraphEditor.DrawPanel, 116
caps.GraphEditor.EdgeProperties, 125
caps.GraphEditor.Editor, 128
caps.GraphEditor.EditorMenuBar, 132
caps.GraphEditor.ExitEditor, 133
caps.GraphEditor.FontConstants, 134
caps.GraphEditor.GE_EditMenu, 135
caps.GraphEditor.GE_FileMenu, 137
caps.GraphEditor.GE_HelpMenu, 139
caps.GraphEditor.GE_PSDLMenu, 141
caps.GraphEditor.GE_ViewMenu, 143
caps.GraphEditor.IdListEditor, 145
caps.GraphEditor.Popup, 148
caps.GraphEditor.PrintJob, 150
caps.GraphEditor.StatusBar, 152
caps.GraphEditor.TextEditEditor153
caps.GraphEditor.ToolBar, 156
caps.GraphEditor.TreePanel, 158
caps.GraphEditor.TreePanelRenderer, 160
caps.GraphEditor.VertexProperties, 163
caps.Psdl.DataFlowComponent, 169
caps.Psdl DataTypes, 174
caps.Psdl.Edge, 176
caps.Psdl.External, 182
caps.Psdl.PSDLTime, 183
caps.Psdl.Vertex, 186
Package Index

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

Package caps

Class Index

- **Caps**
- **EditorDriver**
Package caps.CAPSMain

Class Index

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

Package caps.Display

Class Index

- DisplayComponent
- DisplayExternal
- DisplayVertex
- EdgePath
Package caps.GraphEditor

Class Index

- ColorConstants
- DrawPanel
- EdgeProperties
- Editor
- EditorMenuBar
- ExitEditor
- FontConstants
- GE_EditMenu
- GE_FileMenu
- GE_HelpMenu
- GE_PSDLMenu
- GE_Viewer
- IdListEditor
- Popup
- PrintJob
- StatusBar
- TextEditor
- ToolBar
- TreePanel
- TreePanelRenderer
- VertexProperties
Package caps.Psdl

Class Index

- DataFlowComponent
- DataTypes
- Edge
- External
- PSDL.Time
- Vertex

Class caps.Caps

java.lang.Object

public class Caps
extends java.lang.Object

The driver program for CAPS.

Constructor Index

-Caps()

Method Index

-main(String[])

The constructor 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.EditorDriver

java.lang.Object

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

- 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.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

public CAPSMainMenuBar(CAPSMainWindow owner)

The constructor for this class.

Parameters:

owner - The parent class which has declared this menubar.

Class caps.CAPSMainWindow

public class CAPSMainWindow

extends javax.swing.JFrame

The main CAPS window.

Variable Index

• openPrototypes

The Vector that holds references to the open prototypes
**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 'save required'.

- 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:

ture if none of the prototypes need saving.
public class DatabasesMenu
extends javax.swing.JMenu
implements java.awt.event.ActionListener
This class holds the 'Databases' menu items.

*designDBMenuItem*
Initialize the 'Design Database' event

*swBaseMenuItem*
Initialize 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
Initialize the 'Design Database' event

*swBaseMenuItem*
private javax.swing JMenuItem swBaseMenuItem
Initialize the 'Software Base' event
Constructors

public DatabasesMenu()

Constructor for this class.

Methods

public void actionPerformed(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

caps.CAPSMain.EditMenu

public class EditMenu

extends javax.swing.JMenu

implements java.awt.event.ActionListener

This class holds the 'Edit' menu items.
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

private javax.swing.JMenuItem capsDefaultsMenuItem
    Initiates the 'CAPS Defaults' event

private javax.swing.JMenuItem hwModelMenuItem
    Initiates the 'Hardware Model' event

protected CAPSMainwindow owner
    The main window which owns this menu.

Constructors

public EditMenu(CAPSMainwindow w)
    The constructor for this class.

Parameters:

    w - The parent class which has declared this menubar.

Methods

public void actionPerformed(java.awt.event.ActionEvent e)
    Action event handler for the menu events.
All Packages  Class Hierarchy  This Package  Previous  Next  Index

**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`
private javax.swing.JMenuItem translateMenuItem
    Initiates the 'Translate' event

private javax.swing.JMenuItem scheduleMenuItem
    Initiates the 'Schedule' event

private javax.swing.JMenuItem compileMenuItem
    Initiates the 'Compile' event

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

Class caps.CAPSMain.ExitCAPSMain

java.lang.Object
    +--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.
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
Constructor Index

- `HelpMenu()`
  Constructor for this class.

Method Index

- `actionPerformed(ActionEvent)`
  Action event handler for the menu events.

Constructors

- `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.

This class implements the 'Help' 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.

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.

- processNewItem()

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
  ```java
  private javax.swing.JMenuItem newMenuItem
  ```
  Initiates the 'New' event

- openMenuItem
  ```java
  private javax.swing.JMenuItem openMenuItem
  ```
  Initiates the 'Open' event

- commitWorkMenuItem
  ```java
  private javax.swing.JMenuItem commitWorkMenuItem
  ```
  Initiates the 'Commit Work' event

- retrieveMenuItem
  ```java
  private javax.swing.JMenuItem retrieveMenuItem
  ```
  Initiates the 'Retrieve From DDB' event

- quitMenuItem
  ```java
  private javax.swing.JMenuItem quitMenuItem
  ```
  Initiates the 'Quit' event

- ownerWindow

protected CAPSMainwindow ownerWindow

The main window which owns this menu.

### Constructors

- PrototypeMenu
  ```java
  public PrototypeMenu(CAPSMainwindow owner)
  ```
  Constructor for this class.

**Parameters:**

- owner - The main window which has created this menu.

### Methods

- actionPerformed
  ```java
  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

- processNewItem
  ```java
  public void processNewItem()
  ```
  Handles the event which is caused by selecting the 'New' menu item.

- processOpenMenuItem
  ```java
  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:**
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.

**Variable Index**

- `external`
  
  The External object that is associated with this object.

- `shape`
  
  The shape of the External.

**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.
**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.
Returns:
the shape that represents the External.

Overloads:

getShape in class DisplayComponent

**delete**

public void delete()

Deletes the external that is associated with this object.

Overrides:

delete in class DisplayComponent

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.
**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 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

- **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

- **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 DisplayComponent

- **getInnerShape**
  public java.awt.Shape getInnerShape()
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**

```java
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 this object.

**delete**

```java
public void delete()
```

Deletes the vertex that is associated with this object.

**Overrides:**

delte 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.

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:

gtShape in class DisplayComponent
containsClickedPoint

public boolean containsClickedPoint(int xLoc, int yLoc)

Checks whether the bounding box of the shape contains 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 point of 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
Class `caps.GraphEditor.ColorConstants`

```java
java.lang.Object
    \--- caps.GraphEditor.ColorConstants
```

public class `ColorConstants`

extends java.lang.Object

### Variables

- `COLOR_NAMES`
  ```java
  public static java.lang.String[] COLOR_NAMES
  ```

- `RGB_VALUES`
  ```java
  public static int[] RGB_VALUES
  ```

### Constructors

- `ColorConstants`
  ```java
  public ColorConstants()
  ```
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
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()

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
- 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)
• setSelectionDefault(boolean)

• setSelectionMode(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
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

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

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.

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

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

Parameters:

e - The MouseEvent that occurs.

public void showProperties(DisplayComponent d)

public void setMenuBarItems()

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

Parameters:

e - The MouseEvent that occurs.

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.

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.

public void mouseEntered(java.awt.event.MouseEvent e)
this panel

**Parameters:**

- **e** - The MouseEvent that occurs.

**mouseDragged**

```java
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**

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

**getDiagonalPoint**

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

**mouseMoved**

```java
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**

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

**deleteSelectedComponent**

```java
public void deleteSelectedComponent()
```

**getParentVertex**

```java
public Vertex getParentVertex()
```

**setSelectAllMode**

```java
public void setSelectAllMode(boolean b)
```

**selectAllComponents**

```java
public void selectAllComponents()
```

**setSelectionDefault**

```java
public void setSelectionDefault(boolean b)
```

**setCurrentColor**

```java
public void setCurrentColor(int colorIndex)
```

**setCurrentFont**

```java
public void setCurrentFont(int fontIndex)
```

**rubberBandLine**

```java
protected void rubberBandLine(int x, int y)
```
public class EdgeProperties
extends javax.swing.JDialog
implements java.awt.event.ActionListener
Variables

- targetEdge
- ePath
- nameField
- streamTypeField
- latencyField
- initialValueField
- noButton
- yesButton
- latencyUnitsCombo

Constructor Index

- Edge Properties(Editor)

Method Index

- actionPerformed(ActionEvent)
- getUnitsCombo()
- initialize()
- setEdge(Edge)
- setEdgePath(EdgePath)
- showErrorDialog(String)
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(ActionEvent e)

- showErrorDialog
public void showErrorDialog(String str)

- getUnitsCombo
public javax.swing.JComboBox getUnitsCombo()
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.
**Method Index**

- `checkSaved()`
- `get DataTypes()`
- `getDrawPanel()`
  
  *** Pending -- is it needed? *** Returns the DrawPanel object in this frame
- `getPrototypeFile()`
- `isSaveRequired()`
- `run()`
- `savePrototype()`

**Constructor Index**

- `Editor(File, Vertex, DataTypes)`
  
  The constructor for the editor frame
private final int INITIAL_WIDTH
    The initial width of the GraphEditor

private final int INITIAL_HEIGHT
    The initial height of the Graph Editor

protected DataTypes types

protected java.io.File prototypeFile

protected boolean saveRequired

protected TreePanel treePanel
    The panel that includes the tree structure to view

protected DrawPanel drawPanel
    The panel that the drawing operations are performed.

protected StatusBar statusBar

protected ToolBar tBar
    the main toolbar of the GraphEditor

protected Vertex root

public Editor(java.io.File prototype, Vertex r, DataTypes t)
    The constructor for the editor frame

public void run()

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()
public class EditorMenuBar
extends javax.swing.JMenuBar

The MenuBar of the Graph Editor.

Constructors

public EditorMenuBar (Editor parent)

The constructor for this class.

Constructor Index

- EditorMenuBar (Editor)

The constructor for this class.
Class caps.GraphEditor.ExitEditor

class ExitEditor
extends java.awt.event.WindowAdapter

Closes the caps main window and exits from the program.

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:
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
public static java.lang.String[] FONT_VALUES

public static java.lang.String[] FONT_NAMES

Constructors

FontConstants

public FontConstants()

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.JMenuItem
|      |     |     |      |      |      | +-----
|      |     |     |      |      |      |     | caps.GraphEditor.GE_EditMenu

public class GE_EditMenu

extends javax.swing.JMenuItem

implements java.awt.event.ActionListener

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

**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
Class caps.GraphEditor.GE_FileMenu

caps.GraphEditor.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.
Variables

• 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

Constructors

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

public void actionPerformed(ActionEvent e)

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

Parameters:

e - The associated ActionEvent

Class caps.GraphEditor.GE_HelpMenu

extends javax.swing.JMenuItem

implements java.awt.event.ActionListener

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

- exceptionsMenuItem
  Initiates the 'Exceptions' event

- operatorsMenuItem
  Initiates the 'Operators' event

- psdlGrammarMenuItem
  Initiates the 'PSDL Grammar' event

- streamsMenuItem
  Initiates the 'Streams' event

- timersMenuItem
  Initiates the 'Timers' event

Constructor Index

- GE_HelpMenu(Editor)
  The constructor for the Help menu

Method Index

- actionPerformed(ActionEvent)
  Handles the menu events that occur when one of the menu items is selected

Variables

- psdlGrammarMenuItem
  private javax.swing JMenuItem psdlGrammarMenuItem

- operatorsMenuItem
  private javax.swing JMenuItem operatorsMenuItem

- streamsMenuItem
  private javax.swing JMenuItem streamsMenuItem

- exceptionsMenuItem
  private javax.swing JMenuItem exceptionsMenuItem

- timersMenuItem
  private javax.swing JMenuItem timersMenuItem

Constructors

- GE_HelpMenu
  public GE_HelpMenu(Editor e)

  The constructor for 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

```
javax.swing.JMenuItem
  +----javax.swing.JMenuItem
  
javax.swing.JMenu
  +----javax.swing.JMenuItem
  
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.
**Variable Index**

- `decomposeMenuItem`
  
  Initiates the 'Decompose' event

- `gotoParentMenuItem`
  
  Initiates the 'Goto Parent' event

- `gotoRootMenuItem`
  
  Initiates the 'Goto Root' event

- `parent`
  
  `private Editor parent`  

**Constructor Index**

- `GE_PSDLMenu(Editor)`
  
  The constructor for the PSDL menu

**Method Index**

- `actionPerformed(ActionEvent)`
  
  Handles the menu events that occur when one of the menu items is selected

**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(Editor)`
  
  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
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

• colorMenuItem
  Initiates the 'Color' event

• fontMenuItem
  Initiates the 'Font' event

• manager

• parentFrame

• refreshMenuItem
  Initiates the 'Refresh' event

• selectionModeMenuItem

• toolTipsMenuItem

• treeViewMenuItem
  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

private javax.swing.JMenuItem colorMenuItem

Initiates the 'Color' event

private javax.swing.JMenuItem fontMenuItem

Initiates the 'Font' event

private javax.swing.JMenuItem refreshMenuItem

Initiates the 'Refresh' event

private javax.swing.JCheckBoxMenuItem treeViewMenuItem

Initiates the 'Tree View' event

private javax.swing.JMenuItem toolTipsMenuItem

private javax.swing.JMenuItem selectionModeMenuItem

private javax.swing.JTextField manager

private javax.swing.ToolTipManager manager

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
Class `caps.GraphEditor.IdListEditor`

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

public class `IdListEditor`

extends `java.lang.Object`

implements `java.awt.event.ActionListener`

```
- editButton
- HEIGHT
- helpButton
- inputArea
- model
- okButton
- parentFrame
- promptLabel
- south
- dialog
- vector
```
Constructor Index

AdListEditor(Editor)

Method Index

• actionPerformed(ActionEvent)

• getList()  
• initialize()

• openDialog(Vector)

• setListElements()

• showDialog(String)

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
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.JTextField 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)
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
```

Constructor Index

- Popup(DrawPanel)

Method Index

- getColorMenuItem()
- getDecomposeMenuItem()
- getDeleteMenuItem()
Constructors

- `Popup`
  ```java
class Popup
public Popup(DrawPanel parent)
```
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()
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)

All Packages  Class Hierarchy  This Package  Previous
Next  Index
public class StatusBar
extends javax.swing.JLabel
implements java.awt.event.MouseListener

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)

Class caps.GraphEditor.TextEditr

java.lang.Object
    +----caps.GraphEditor.TextEditr

public class TextEditor
extends java.lang.Object
implements java.awt.event.ActionListener

Variable Index

- allowsEmptyString
- cancelButton
- dialog
- grammarKind
Constructor Index

- TextEditor(Editor)

Method Index

- actionPerformed(ActionEvent)
- getString()
- initialize()
- openDialog(String, String, String, int, boolean)
- showErrorMessage(String)

Variables

- dialog
  private static javax.swing.JDialog dialog
- south
  private static javax.swing.JPanel south
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)

Constructors

• TextEditor
Class caps.GraphEditor.ToolBar

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

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.

- 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
- ToolBar(Editor)
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:
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

Class caps.GraphEditor.TreePanel

caps.GraphEditor.TreePanel

caps.GraphEditor.TreePanel

caps.GraphEditor.TreePanel

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.
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)

- treeNodeRemoved(TreeModelEvent)

- treeNodeInserted(TreeModelEvent)

- valueChanged(TreeSelectionEvent)

- structureChanged(TreeModelEvent)

- view политик(TreeModelEvent)

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.
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)`
- 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(java.awt.JTree tree,
java.lang.Object value,
selected,
expanded,
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:

paint in class javax.swing.JComponent
public class VertexProperties
extends javax.swing.JDialog
implements java.awt.event.ActionListener
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
public void setButtonText(javax.swing.JButton b, java.lang.Object o)

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)

- showAlertDialog
  public void showAlertDialog(java.lang.String str)

- setButtonText
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.
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.

- getFont()

  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.

- getMetFont()

  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.
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.

---

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**
protected PSDL.Time 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.

● met1Font
protected java.awt.Font met1Font

The font representation of the met (or latency).

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 value.

Constructors

● DataFlowComponent
protected DataFlowComponent(Widget v)

The constructor for this class.

Parameters:

v - The parent vertex of this component
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.

**getFont**

public java.awt.Font getFont()

    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.

**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.

**getMetFont**

public java.awt.Font getMetFont()

    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**
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.
Constructor Index

- DataTypes()
- isPredefined(String)
- removeElements()
- toString()

Method Index

- addType(String)
- addType(String, String, String)
- build(StreamTokenizer)
- buildTypes(File)
- buildTypes(String)
- exists(String)
- getNextToken(StreamTokenizer)

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
public void addType(java.lang.String name, java.lang.String spec, java.lang.String impl)

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

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

destination

The destination Vertex of this stream.

initialValue

The initial value of the stream.
**Constructor Index**

- `Edge(int, int, Vertex)`
  The constructor for this class.

**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()`
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.

• isStreamState()
  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.

• setStreamStream(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
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.

## Constructors

*Edge*

```java
public Edge(int xLocation,
            int yLocation,
            Vertex v)
```

The constructor for this class.

**Parameters:**

v - the parent vertex of this edge.

## Methods

*moveTo*

```java
public void moveTo(int xOffset,
                   int yOffset)
```

Relocates the stream when the stream is moved with other objects.

**Overrides:**

moveto in class **DataFlowComponent**

*reShape*

```java
public void reShape(int xLocation,
                    int yLocation)
```

Is called when one of the handles of the stream is dragged in the DrawPanel.

*setX*

```java
public void setX(int newX)
```
Changes the x value of the stream to the specified value.

```java
public void setX(int newx)
Changes the y value of the stream to the specified value.

public void setY(int newY)
Changes selectedHandleIndex to the specified value.

public int getSelectedHandleIndex(int i)

getSelectedHandleIndex

Overrides:
- getX in class DataFlowComponent
- getY in class DataFlowComponent
```

Returns the x value of this stream.

```java
public int getX()

getX

Overrides:
- getY in class DataFlowComponent
```

Returns the y value of this stream.

```java
public int getY()

getY

Overrides:
- getX in class DataFlowComponent
- getY in class DataFlowComponent
```

Returns the type of this stream.

```java
public java.lang.String getStreamType()

getStreamType

Sets the type of this stream to the specified value.

public void setStreamType(java.lang.String type)

setStreamType

Sets the destination Vertex of this stream to the specified value.

```java
public void setDestination(Verte v)

setDestination

Returns the destination Vertex of this stream.

public Vertex getDestination()

getDestination

Returns true if this is a state stream.

public boolean isStateStream()

isStateStream

Changes the isState field to the specified value.

```java
public void setStateStream(boolean flag)

setStateStream

Returns the initial value of this stream.

public java.lang.String getInitialValue()

getInitialValue

Sets the initial value of this stream to the specified value.

public void setInitialValue(java.lang.String str)

setInitialValue
• **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 control point.
  y - the y component of the new control 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()
  
  Deletes this stream.

• **deleteHelper**
  
  public void deleteHelper()
  
  Helper method to delete the stream.
Constructors

public External(int xLocation, 
               int yLocation, 
               Vertex v)

Methods

public java.awt.Point getIntersectionPoint(java.awt.Point p)

Overrides:

getIntersectionPoint in class Vertex
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 milliseconds.

• 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 value of this object.
- **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 milliseconds.

- **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 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:
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 initial radius of a Vertex.

• **keywordList**

• **mcp**

• **mcpReqmts**

• **metReqmts**

• **mrt**

• **mrtReqmts**

• **NON_TIME_CRITICAL**
   The constant value for non-time critical 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()
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.
•getImplLanguage()
  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.
•getMrtReqmts()
  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.

*setImplLanguage(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 requirements to the specified value.

*setTerminator(boolean)
Sets this component as a terminator or a stream.

- `setTimerList(Vector)`
  Sets the timer list to the specified value.
- `setTimerOpList(String)`
  Sets the timer op list to the specified value.
- `setTimingType(int)`
  Sets the timing type to the specified value.
- `setTriggerReqs(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.

## Variables

- **INITIAL_RADIUS**
  public static final int `INITIAL_RADIUS`
  The constant value for the initial radius of a Vertex.

- **NON_TIME_CRITICAL**
  public static final int `NON_TIME_CRITICAL`
  The constant value for non-time critical 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 construct.
true if this Vertex is a terminator.

The x-location of this component on the DrawPanel

The y-location of this component on the DrawPanel

The width of this component. It serves as the radius of an operator.

The width of a terminator. width of operator.

cap

The height of this component.

color

The color parameter of this component.

color

The color parameter of this component.

protected java.util.Vector triggerReqs

protected java.util.Vector triggerStreamList

protected java.lang.String triggerCondition

protected java.lang.String outputGuardList

protected java.util.Vector mrt

protected java.util.Vector mcp

protected PSDDTime mrt

protected PSDDTime mcp

finish Within

FinishWithin

FinishWithinReqs
```
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
```
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

public boolean isTerminator()

        Returns true if this component is a terminator.

Returns:

        true if this component is a terminator.

public void setTerminator(boolean b)

        Sets this component as a terminator or a stream. Also changes the
        width of the component.

Parameters:

    b -

public void correctInOutStreams()

        Corrects the ending points of the in and out streams of this
        component.

public int getWidth()

        Returns the width of this component.

Returns:

        the width of this component.

public void setWidth(int w)

        Changes the width of this component.

Parameters:

    w - the new width of this component.

public int getHeight()

        Returns the height of this component.

Returns:

        the height of this component.

public int getX()

        Returns the x component of the location of this Vertex

Returns:

        x

Overrides:

    getX in class DataFlowComponent
public void setX(int xLoc)

Changes the x component of the location of this Vertex.

Parameters:
  xLoc.

public void setY(int yLoc)

Changes the y component of the location of this Vertex.

Parameters:
  yLoc.

public int getY()

Returns the y component of the location of this Vertex.

Returns:
  y

Overrides:
  getY in class DataFlowComponent

public void setColor(int c)

Changes the color value for this Vertex.

Parameters:
  c - the new color value.

public int getColor()

Returns the color value for this Vertex.

Returns:
  the color value of the Vertex.

public void addInEdge(Edge e)

Adds a new Edge to the inEdges Vector.

Parameters:
  e - the new inEdge.

public void removeInEdge(Edge e)

Removes an Edge from the inEdges Vector.

Parameters:
  e - the inEdge to be removed.

public void addOutEdge(Edge e)

Adds a new Edge to the outEdges Vector.

Parameters:
  e - the new outEdge.

public void removeOutEdge(Edge e)
Removes an Edge from the outEdges Vector.

**Parameters:**

- *e* - the outEdge to be removed.

**getTimingType**

```java
public int getTimingType()
```

Returns the timing type of this Vertex.

**setTimingType**

```java
public void setTimingType(int type)
```

Sets the timing type to the specified value.

**getTriggerType**

```java
public int getTriggerType()
```

Returns the triggering type of this Vertex.

**setTriggerType**

```java
public void setTriggerType(int type)
```

Sets the triggering type to the specified value.

**getPeriod**

```java
public PSDLTime getPeriod()
```

Returns the period value of this Vertex.

**getFinishWithin**

```java
public PSDLTime getFinishWithin()
```

Returns the finish within value of this Vertex.

**getMcp**

```java
public PSDLTime getMcp()
```

Returns the mcp value of this Vertex.

**getMrt**

```java
public PSDLTime getMrt()
```

Returns the mrt value of this Vertex.

**setPeriod**

```java
public void setPeriod(PSDLTime p)
```

Sets the period to the specified value.

**setFinishWithin**

```java
public void setFinishWithin(PSDLTime fw)
```

Sets the finish within to the specified value.

**setMcp**

```java
public void setMcp(PSDLTime m)
```

Sets the mcp to the specified value.

**setMrt**

```java
public void setMrt(PSDLTime mr)
```

Sets the mrt to the specified value.

**getImplLanguage**

```java
public java.lang.String getImplLanguage()
```

Returns the implementation language of this Vertex.

**setImplLanguage**

```java
public void setImplLanguage(java.lang.String s)
```

Sets the implementation language to the specified value.

**getMetReqmts**

```java
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 requirements 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()

- 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)

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

caps.Caps, 202

caps.CAPSMain.CAPSMainMenuBar, 202
caps.CAPSMain.CAPSMainWindow, 203
caps.CAPSMain.DataBasesMenu, 205
caps.CAPSMain>EditMenu, 206
caps.CAPSMain.ExecSupportMenu, 207
caps.CAPSMain.ExitCAPSMain, 208
caps.CAPSMain.HelpMenu, 209
caps.CAPSMain.PrototypeMenu, 209

caps.Display.DisplayComponent, 212
caps.Display.DisplayExternal, 214
caps.Display.DisplayVertex, 215
caps.Display.EdgePath, 217

caps.GraphEditor.ColorConstants, 219
caps.GraphEditor.DrawPanel, 219
caps.GraphEditor.EdgeProperties, 231
caps.GraphEditor.Editor, 233
caps.GraphEditor.EditorMenuBar, 236
caps.GraphEditor.ExitEditor, 237
caps.GraphEditor.FontConstants, 237
caps.GraphEditor.GE_EditMenu, 238
caps.GraphEditor.GE_FileMenu, 239
caps.GraphEditor.GE_HelpMenu, 241
caps.GraphEditor.GE_PSDLMenu, 242
caps.GraphEditor.GE_ViewMenu, 243
caps.GraphEditor.IdListEditor, 245
caps.GraphEditor.Popup, 247
caps.GraphEditor.PrintJob, 248
caps.GraphEditor.StatusBar, 250
caps.GraphEditor.TextEditor, 250
caps.GraphEditor.ToolBar, 252
caps.GraphEditor.TreePanel, 254
caps.GraphEditor.TreePanelRenderer, 256
caps.GraphEditor.VertexProperties, 257
caps.Psdl.DataFlowComponent, 265
caps.Psdl.DataTypes, 269
caps.Psdl.Edge, 271
caps.Psdl.External, 275
caps.Psdl.PSDLTime, 276
caps.Psdl.Vertex, 277

201
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
package caps.CAPSMain;

import java.awt.*;
import java.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.List;

/**
 *   * The main CAPS window.
 * @author Ilker DURANLI OGLU
 * @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;
    }
/**
 * 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
        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);
        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
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
package caps.CAPSMain;

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 JMenuItem 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("CAPS Defaults");

/** *
 * 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");
    }
package caps.CAPSMain;
import javax.swing.JMenuItem;
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);
    }
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
package caps.CAPSMain;

import javax.swing.JMenuItem;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * 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

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.
     */
    public PrototypeMenu()
    {
        super ();
    }
    public void actionPerformed(ActionEvent e)
    {
        // Not implemented yet
    }
    private JMenuItem newMenuItem = new JMenuItem ("New");

    private JMenuItem openMenuItem = new JMenuItem ("Open");

    private JMenuItem commitWorkMenuItem = new JMenuItem ("Commit Work");

    private JMenuItem retrieveMenuitem = new JMenuItem ("Retrieve From DDB");

    private JMenuItem quitMenuItem = new JMenuItem ("Quit");

    protected CAPSMainWindow ownerWindow;

    public PrototypeMenu()
    {
        super ();
    }
    public void actionPerformed(ActionEvent e)
    { // Not implemented yet
    }
}
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);
    } else if (e.getActionCommand () == "processNewMenuItem") {
        processNewMenuItem ();
        if (protoHome == null) { // If it is not set as a command line
            String protoHome = System.getProperty ("PROTOTYPEHOME");
            File protoDir;
            if (protoHome == null) {
                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 {
            File file = new File (protoHome + File.separator + proto +
                File.separator + name + ".psdl1");
            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);
                    }
                }
            } else {
                File deleteFile = new File (protoHome + File.separator + proto +
                File.separator + name + ".psdl1");
                try {
                    deleteFile.delete ();
                } catch (IOException ex) {
                    System.out.println ("Could not delete file: \n    " + deleteFile.getCanonicalPath () + "\n    " + ex.getMessage ()�");
                }
            }
        }
    }
}
---

```java
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.getAbsoluteFile () +
                                File.separator +
                                selectedDir.getParentFile ().getName () + ".psdl");
            if (!file.exists ()
                JOptionPane.showMessageDialog (ownerWindow, "The selected prototype file cannot be opened", "Error Message",
                       JOptionPane.ERROR_MESSAGE);
            }
        }

    }

}
```
package caps.Display;

import java.awt.*;
import java.awt.font.*;
import java.awt.geom.*;
import javax.swing.JLabel;
import java.util.Vector;

/**
 * This is an abstract super class of EdgePath and DisplayVertex.
 * @author Ilker DURANLI OGLU
 * @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.getFont (),
        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;
    }

    /**
* Gets the location of the label shape and draws it into 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.getFontFont (), g2D.getFontRenderContext ());
    else
      metShape = new TextLayout (* *, dfc.getFontFont (),
      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) { // It may not have an met
      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
package caps.Display;
import java.awt.geom.*;
import java.awt.*;
import java.awt.font.*;
import caps.Pndl.*;
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.
 */
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.
 */
public boolean containsClickedPoint (int xLoc, int yLoc)
{
    return false;
}

/**
 * Returns the vector that contains the handles of the shape.
 */
public Vector getHandles ()
{
    Vector v = new Vector ();
    return v;
}

/**
 * Returns 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
package caps.Display;

import java.awt.geom.*;
import java.awt.*;
import java.awt.font.*;
import caps.Pslcl.*;
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.getMinX () - i, s.getMinY () - i,
                                HANDLESIZE, HANDLESIZE));
  v.add (new Rectangle2D.Double (s.getMinX () - i, s.getMinY () - i,
                                HANDLESIZE, HANDLESIZE));
  v.add (new Rectangle2D.Double (s.getMinX () - i, s.getMinY () - i,
                                HANDLESIZE, HANDLESIZE));
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.
 */
public Shape getShape ()
{
    return (Shape) shape;
}

/**
 * This method is called if the Vertex is composite. It calculates and returns
 * a smaller inner shape.
 */
public Shape getInnerShape ()
{
    if (vertex.isTerminator ())
        return (Shape) new Rectangle2D.Double (shape.getX () + 4,
shape.getY () + 4, shape.getHeight () - 8, shape.getWidth () - 8);
    else
        return (Shape) new Ellipse2D.Double (shape.getX () + 4,
shape.getY () + 4, shape.getHeight () - 8, shape.getWidth () - 8);
}

/**
 * The shape that is returned will be painted with the color of the Vertex.
 */
@Override
public Shape getPaintedShape ()
{
    if (vertex.isTerminator ())
        return (Shape) new Rectangle2D.Double (shape.getX () + 1,
shape.getY () + 1, shape.getHeight () - 1.0f, shape.getWidth () - 1.0f);
    else
        return (Shape) new Rectangle2D.Double (shape.getX () + 5,
shape.getY () + 5, shape.getHeight () - 10, shape.getWidth () - 10);
    else {
        if (vertex.isLeaf ()
            return (Shape) new Ellipse2D.Double (shape.getX () + 1f,
shape.getY () + 1f, shape.getHeight () - 2f, shape.getWidth () - 2f);
        else
            return (Shape) new Ellipse2D.Double (shape.getX () + 5,
shape.getY () + 5, shape.getHeight () - 10, shape.getWidth () - 10);
    }
}

/**
 * Deletes the vertex that is associated with this object.
 */
public void delete ()
{
    vertex.delete ();
    vertex = null;
    shape = null;
}

} // End of the class DisplayVertex
package caps.Display;

import caps.Edl.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 DURANLOGLU
 * @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);
/**
 * 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.
 */
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
package caps.GraphEditor;

public class ColorConstants {


    public static int RGB_VALUES [] = { 7396243, 0, 255, 10444703, 10889770, 6267973, 16744192, 4342383, 65535, 3100463, 5197615, 10040013, 7021454, 3100495, 7377883, 5526612, 9315107, 2330147, 13667442, 14408560, 12632256, 65280, 9689968, 5123887, 10461023, 12638681, 11053224, 9408445, 3329330, 16711935, 9315179, 3329433, 3289005, 9662683, 4353858, 8323327, 8388352, 7396315, 14381203, 3092303, 2302862, 16744192, 16720896, 14381275, 941919, 12357519, 15379946, 1671680, 7891948, 2330216, 9333539, 331672, 32767, 6507, 2321294, 14390128, 14204888, 11397866, 5189555, 13382297, 14211263, 16777215, 16776960, 10079282 ];

} // End of the class ColorConstants

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.Psd1.*;
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;
/**
 * 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
 */
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
    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;
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 ().setAllowChildren
            (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 () &&
                    !((Vertex) dfc).isTerminal) 
                    DisplayVertex op = new DisplayVertex ((Vertex) dfc);
                    op.setLabelShape (g2D);
                    op.setMetaShape (g2D);
                    displayComponentVector.addElement (op);
                }
        else if (dfc instanceof Vertex && (((Vertex) dfc).isTerminator
            () &&
            !((Vertex) dfc).isTerminal)) 
            DisplayVertex tr = new DisplayVertex ((Vertex) dfc);
            tr.setLabelShape (g2D);
            tr.setMetaShape (g2D);
            displayComponentVector.addElement (tr);
        else {
            EdgePath ep = new EdgePath ((Edge) dfc);
            ep.setLabelShape (g2D);
            ep.setMetaShape (g2D);
            ((Edge) ep.getDataFlowComponent ()).correctLabelOffset ();
        }
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.
 */
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.setNetFontIndex (currentFont + 1);
    parentFrame.getTreePanel ().addNewDFC (op, parentVertex);
    displayVertex opCircle = new DisplayVertex (op);
    opCircle.setLabelShape (g2D);
    opCircle.setMetaShape (g2D); // *** Maybe we don't need this
    *****
    paintComponent (opCircle);
    displayComponentVector.addElement (opCircle);
}

// ********* Pending drawlist ?? **********
/**
 * @param component OPERATOR, TERMINATOR or STREAM
 */
* 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);
* parentFrame.getTreePanel ().addNewDFC (term, parentVertex);
* DisplayVertex trectangle = new DisplayVertex (term);
* trectangle.setLabelShape (g2D);
* tRectangle.setMetaShape (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.addElement ((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.setMetaShape ((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.setMetaFontIndex (currentFont + 1);
ex.setLabelFontIndex (currentFont + 1);
ex.setMetaFontIndex (currentFont + 1);
ex.addOutEdge (ed);
ed.setSource (ex);
currentEdge = new EdgePath (ed);
DisplayExternal extern = new DisplayExternal (ex);
extern.setLabelShape ((Graphics2D) getGraphics () ;
extern.setMetaShape ((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.setMetaFontIndex (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
* */
public void paintComponent (DisplayComponent component)
{
    Graphics2D g2D = (Graphics2D) getGraphics () ;
    if (component instanceof DisplayVertex) {
        g2D.setColor (new Color (ColorConstants.RBG_VALUES [currentColor]));
        g2D.fill (((DisplayVertex) component).getPaintedShape ());
        g2D.fill (component.getShape ());
        g2D.drawRect (Color.black);
    } else {
        g2D.draw (component.getShape ());
        component.drawLabelShape (g2D);
        if (component instanceof DisplayVertex) {
            component.drawNetShape (g2D);
            g2D.draw (component.getShape ());
        }
    }
}

/**
 * This method is called to repaint all the components when necessary.
 */
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 dcf = dcp.getDataFlowComponent ();
        if ((MOVING_COMPONENT ||
        if (dcp instanceof DisplayVertex)
        {
            g2D.setColor (new Color (ColorConstants.RBG_VALUES [(Vertex) dcf].getColor () - 1));
            g2D.fill (((DisplayVertex) dcp).getPaintedShape ());
            g2D.setShape (Color.black);
            g2D.draw (dcp.getShape ());
        }
        else {
            if (dcp instanceof DisplayVertex)
            {
                g2D.setColor (new Color (ColorConstants.RBG_VALUES [(Vertex) dcf].getColor () - 1));
                g2D.fill (((DisplayVertex) dcp).getPaintedShape ());
                g2D.setShape (Color.black);
                g2D.draw (dcp.getShape ());
            }
        }
    }
}

if (!dcf.isLeaf ())
    g2D.draw ((DisplayVertex) dcp).getInnerShape ();
}

dcp.drawLabelShape (g2D);
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
 */
public Dimension getPreferredSize ()
{
    return new Dimension (WIDTH, HEIGHT);
}

/**
 * Handles the event that occurs when a mouse button is clicked on this panel
 */
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 (isSelectAllMode & isHoldingHandle (xPosition, yPosition))
        {
            System.out.println ("Selecting Handles");
            if (selectedComponent instanceof DisplayVertex &
            selectedComponent.getShape ().contains (diagonalPoint)) // Make sure it is not the label
            { RESIZING = true;
            }
        }
    }
}
else if (selectMode) {
    // 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)) {
            if (dc.getShapeBounds ().contains (xPosition, yPosition)) {
                System.out.println (selectedComponent + " Moving_All");
                MOVING_ALL = true;
                flag = true;
            } else if (dc.getLabelShapeBounds ().contains (xPosition, yPosition)) {
                System.out.println ("Inside the label");
                eraseHandles (); // ***** Pending this is too simple
                handlesVector = dc.getStringHandles (dc.getShapeBounds () ;
                selectedComponent = dc;
                paint (getGraphics () ;
                flag = true;
            } else if (dc.getShapeBounds () .contains (xPosition, yPosition)) {
                System.out.println ("Inside the met");
                eraseHandles (); // ***** Pending this is too simple
                handlesVector = dc.getStringHandles (dc.getShapeBounds () ;
                selectedComponent = dc;
                paint (getGraphics () ;
                flag = true;
            } // **** PENDING *****
            else if (dc.containsClickedPoint (xPosition, yPosition)) {
                System.out.println ("Yes, inside");
                eraseHandles (); // This is more work can look at it in the older versions
                handlesVector = dc.getHandles () ;
                selectedComponent = dc;
                paint (getGraphics () ; // ***** Pending calls the paint method twice here
                flag = true; // Clicked on a component
            }
        }
    }

    if (flag)
        setCursor (MOVE_CURSOR);
    }

    if (selectedComponent != null && !flag) { // Clicked on an empty area
        selectedComponent = null; // I put it here
        not to call erase handles 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 chris' implementation
                parentFrame.setSaveRequired (true);
                break;
                default: break;
            }
        }
    public void setMenuBarItems () {
        if (selectedComponent == null) {
            parentFrame.getJMenuBar () .getItem (1).setEnabled (false); // delete
            parentFrame.getJMenuBar () .getItem (3).setEnabled (false); // decompose
        }
        parentFrame.getJMenuBar () .getItem (2).setEnabled (true); // color
    else { ...
public void mouseEntered(MouseEvent e) {
    //System.out.println("MouseEntered");
}

/**
 * Handles the event that occurs when the mouse exits the panel.
 */
public void mouseExited(MouseEvent e) {
    //System.out.println("MouseExited");
}

/**
 * Handles the event that occurs when a mouse button is pressed on this panel.
 */
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) {
        final External ex = new External(xPosition, yPosition, parentVertex);
        ex.addInEdge((Edge) currentEdge.getDataFlowComponent());
        ((Edge) currentEdge.getDataFlowComponent()).setDestination(ex);
        ((Edge) currentEdge.getDataFlowComponent()).addPoint(xPosition,
            yPosition);
        parentFrame.getTreeNode().addNewDFC((Edge)
            currentEdge.getDataFlowComponent(), parentVertex);
        currentEdge.setLabelShape((Graphics2D) getGraphics());
        currentEdge.setMetaShape((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.setMetaShape((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);
if (selectAllMode && (MOVING_ALL)) {
    //System.out.println ("Moving all");
    handlesVector.removeAllElements ();
    bounds = (Rectangle) (DisplayComponent)
    .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.removeAll (selectedComponent.getHandles ());
        handlesVector.addAll (selectedComponent.getStringHandles
        (selectedComponent.getLabelShapeBounds ()));
        handlesVector.addAll (selectedComponent.getShapeBounds ()
        (selectedComponent.getStringHandles
        (selectedComponent.getShape ()));
        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_MST) {
            dfc.setMstOffset (xPosition - prevPoint.x, yPosition -
            prevPoint.y); handlesVector = selectedComponent.getStringHandles
            (selectedComponent.getShapeBounds ());
        } 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.getShapeBounds ());
        } else if (MOVING_MST) {
            dfc.setMstOffset (xPosition - prevPoint.x, yPosition -
            prevPoint.y); handlesVector = selectedComponent.getStringHandles
            (selectedComponent.getShapeBounds ());
        } 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.getShapeBounds ());
        } else if (MOVING_MST) {
            dfc.setMstOffset (xPosition - prevPoint.x, yPosition -
            prevPoint.y); handlesVector = selectedComponent.getStringHandles
            (selectedComponent.getShapeBounds ());
        } 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.getShapeBounds ());
        } else if (MOVING_MST) {
            dfc.setMstOffset (xPosition - prevPoint.x, yPosition -
            prevPoint.y); handlesVector = selectedComponent.getStringHandles
            (selectedComponent.getShapeBounds ());
        } 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.getShapeBounds ());
        } else if (MOVING_MST) {
            dfc.setMstOffset (xPosition - prevPoint.x, yPosition -
            prevPoint.y); handlesVector = selectedComponent.getStringHandles
            (selectedComponent.getShapeBounds ());
        } 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.getShapeBounds ());
        } else if (MOVING_MST) {
            dfc.setMstOffset (xPosition - prevPoint.x, yPosition -
            prevPoint.y); handlesVector = selectedComponent.getStringHandles
            (selectedComponent.getShapeBounds ());
        } 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.getShapeBounds ());
        } else if (MOVING_MST) {
            dfc.setMstOffset (xPosition - prevPoint.x, yPosition -
            prevPoint.y); handlesVector = selectedComponent.getStringHandles
            (selectedComponent.getShapeBounds ());
        } 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.getShapeBounds ());
        } else if (MOVING_MST) {
            dfc.setMstOffset (xPosition - prevPoint.x, yPosition -
            prevPoint.y); handlesVector = selectedComponent.getStringHandles
            (selectedComponent.getShapeBounds ());
        } 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.getShapeBounds ());
        } else if (MOVING_MST) {
            dfc.setMstOffset (xPosition - prevPoint.x, yPosition -
            prevPoint.y); handlesVector = selectedComponent.getStringHandles
            (selectedComponent.getShapeBounds ());
        } 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.getShapeBounds ());
        } else if (MOVING_MST) {
            dfc.setMstOffset (xPosition - prevPoint.x, yPosition -
            prevPoint.y); handlesVector = selectedComponent.getStringHandles
            (selectedComponent.getShapeBounds ());
        } 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.getShapeBounds ());
        } else if (MOVING_MST) {
            dfc.setMstOffset (xPosition - prevPoint.x, yPosition -
            prevPoint.y); handlesVector = selectedComponent.getStringHandles
            (selectedComponent.getShapeBounds ()):}
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.getLabelShape Bounds () . contains  
    (xPosition, yPosition)) {  
    MOVING_LABEL = true;  
    if (dfc instanceof External) {  
      ((External) dfc). setLocation (xPosition - prevPoint.x,  
      yPosition - prevPoint.y);  
      selectedComponent.update ();  
    }  
    else  
      dfc.setLocation (xPosition - prevPoint.x,  
      yPosition - prevPoint.y);  
    handlesVector = selectedComponent.getStringHandles  
    (selectedComponent.getLabelShape Bounds () );  
    parentFrame.setSaveRequired (true);  
  }  
  else if (selectedComponent.getLabelShape Bounds () . contains  
    (xPosition, yPosition)) {  
    MOVING_MET = true;  
    dfc.setLocation (xPosition - prevPoint.x, yPosition -  
    prevPoint.y);  
    handlesVector = selectedComponent.getStringHandles  
    (selectedComponent.getLabelShape Bounds () );  
    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);  
}  
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 . getMinY () )  
    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 . getMaxY () <=  
    r2D . getMinY () )  
    p . setLocation (r2D . getMaxX () - w, r2D . getMinY () - w);  
  else if (rect . getMinX () <= r2D . getMinX () & & rect . getMaxY () >=  
    r2D . getMinY () )  
    p . setLocation (r2D . getMinX () - w, r2D . getMaxY () + 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 . getEX ();  
  //getGraphics ();  
  setCursor (MOVE_CURSOR);  
  //System . out . println ("Inside mouse moved");  
  return p;  
}
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)) {
            if (dc.equals (selectedComponent) && (MOVING_LABEL || MOVING_WET))
                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 ();
    } else 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.setMetaShape ((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 ());
        setParentVertex (parentVertex, null); // This takes care
            of unremoved streams
        parentFrame.setSaveRequired (true);
    }
}

public Vertex getParentVertex () {
    return parentVertex;
}

public void setSelectAllMode (boolean b) {
    selectAllMode = b;
    if (selectAllMode)
        selectAllComponents ();
}
public void selectAllComponents ()
{
    DisplayComponent dc = null;
    handlesVector.removeAllElements ();
    if (!displayComponentVector.isEmpty () )
        bounds = (Rectangle) 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.getMetaShapeBounds () ));
    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) currentNode.getDataFlowComponent () ).getPoints () .lastElement ();
    Graphics g = getGraphics ();
    g.setColor ( new Color ( 128, 128, 128 ));
    g.setXORMode ( Color.white );
    g.drawLine ( last.x, last.y, x, y );
    g.setPaintMode ();
    g.setColor ( Color.black );
}
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{

    private Edge targetEdge;
    private EdgePath ePath;
    JTextField nameField;
    JTextField streamTypeField;
    JTextField latencyField;
    JTextField initValueField;
    JRadioButton noButton;
    JRadioButton yesButton;
    JComboBox latencyUnitsCombo;
    JButton okButton;
    JButton cancelButton;
    JButton helpButton;
    JButton initValueButton;

    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);
        gbc.gridx = 0;
        gbc.gridy = 0;
        box.add (new JLabel("

        JPanel namePanel = new JPanel (new GridBagLayout());
        namePanel.setBorder (BorderFactory.createTitledBorder ("\n
        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);
        latencyUnitsCombo = getUnitsCombo();
        gbc.gridx = 2; gbc.gridy = 0; gbc.gridy = 0;
        namePanel.add (new JLabel ("Stream Name:", gbc);
        gbc.gridx = 3; gbc.gridy = 2; gbc.gridy = 1;
        namePanel.add (nameField, gbc);
        gbc.gridx = 2; gbc.gridy = 2; gbc.gridy = 1;
        namePanel.add (new JTextField (10), gbc);
        gbc.gridx = 3; gbc.gridy = 2; gbc.gridy = 1;
        namePanel.add (new JLabel ("Stream Type:", gbc);
        gbc.gridx = 2; gbc.gridy = 0; gbc.gridy = 2;
        namePanel.add (new JLabel ("Is it a state stream?", gbc);
        gbc.gridx = 3; gbc.gridy = 3; gbc.gridy = 3;
        namePanel.add (new JLabel ("Latency ", gbc);
        gbc.gridx = 2; gbc.gridy = 2; gbc.gridy = 2;
        namePanel.add (new JLabel ("\n
        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.gridx = 1; gbc.gridy = 1; gbc.gridy = 0;
        okPanel.add (okButton, gbc);
        gbc.gridx = 1; gbc.gridy = 4;
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 ()
    } else if (e.getSource () == noButton) {
        initValueField.setText ("");
        initValueButton.setEnabled (false);
    } else 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 ());
    } else if (e.getSource () == okButton) {
        boolean errorStatus = false;
        String str = nameField.getText ();
        if (!GrammarCheck.isValid (str, GrammarCheck.ID)) {
            showErrorMessage ("Illegal stream name");
            errorStatus = true;
        }
        str = streamTypeField.getText ();
        if (!GrammarCheck.isValid (str, GrammarCheck.TYPE_NAME)) {
            showErrorMessage ("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)) {
                showErrorMessage ("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 ());
        }
    }
targetEdge.setStateStream (true);
)
if (targetEdge.isStateStream ()) {
    Vertex parent = (Vertex) targetEdge.getParent ()
    ((java.util.Vector) parent.getSpecReqsmts ()).elementAt
(2)).addElement ("");
)
ePath.setLabelShape ((Graphics2D) parentFrame.getDrawPanel
().getGraphics ());
ePath.setMetaShape ((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 showErrorMessage (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

package caps.GraphEditor;

import java.awt.*;
import java.awt.event.*;
import java.swing.*;
import java.swing.event.*;
import caps.Pndl.Vertex;
import caps.Pndl DataTypes;
import caps.Parser.*;
import java.io.*;

/**
 * The main frame for the Graph Editor.
 * It constructs and drivers 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
 */
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);

    treeNode = new TreeNode (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 (1 / 5));
    innerSplit.setBorder (BorderFactory.createLoweredBevelBorder ());

    panel.add (tBar, BorderLayout.NORTHEAST);
    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);
/**
 * *** 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 get DataTypes ()
{
    return types;
}

public File getPrototypeFile ()
{
    return prototypeFile;
}

public void setSaveRequired (boolean b)
{
    saveRequired = b;
    if (saveRequired) {
        statusBar.setText ("Save required");
        jMenuBar ().getMenu (0).getItem (0).setEnabled (true); //
        enable save menu item
    }
    else {
        statusBar.setText ("Save not required");
        jMenuBar ().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);
        String writer = new StringWriter ();
        CreatePsd1.ReInit (writer);
        CreatePsd1.build (root, types);
        String str = CreatePsd1.getPsd1 ();
        testFile.write (str);
    }
package caps.GraphEditor;

import javax.swing.*;

/**
 * The MenuBar of the Graph Editor.
 *
 * @author Ilker DURANLI GLOYU
 * @version
 */
public class EditorMenu 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
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
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 Iker DURANLIOGLU
 * @version
 */
public class GE_EditMenu extends JMenuItem 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) {
package caps.GraphEditor;

import javax.swing.*;
import java.awt.event.*;
import java.awt.Cursor;
import caps.Psi1.Vertex;

/**
 * Constructs the File menu of the menubar.
 * Also handles the events associated with the File Menu.
 * @author Iker DURANLIJOGLU
 * @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
}
package caps.GraphEditor;
import javax.swing.*;
import java.awt.event.*;

/**
 * Constructs the Help menu of the menubar.
 * It handles the events associated with the Help Menu.
 * @author Ilker DURANLI OGLU
 * @version
 */
public class GE_HelpMenu extends JMenuItem 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);

    psdlGrammarMenuItem.addActionListener ("Opens help about PSDL Grammar");
    operatorsMenuItem.addActionListener ("Opens help about operators");
    streamsMenuItem.addActionListener ("Opens help about streams");
    exceptionsMenuItem.addActionListener ("Opens help about exceptions");
    timersMenuItem.addActionListener ("Opens help about timers");

    psdlGrammarMenuItem.addMouseLister (e.getStatusBar ());
    operatorsMenuItem.addMouseLister (e.getStatusBar ());
    streamsMenuItem.addMouseLister (e.getStatusBar ());
    exceptionsMenuItem.addMouseLister (e.getStatusBar ());
    timersMenuItem.addMouseLister (e.getStatusBar ());

    psdlGrammarMenuItem.addActionLister (this);
    operatorsMenuItem.addActionLister (this);
    streamsMenuItem.addActionLister (this);
    exceptionsMenuItem.addActionLister (this);
    timersMenuItem.addActionLister (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");
    } else if (e.getSource () == timersMenuItem) 
    { System.out.println ("Timers help has not yet been implemented");
    }
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");
    }
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 TipManager manager;
private Editor parentFrame;

/**
 * The constructor for the View menu
 */

public GE_ViewMenu(Editor parent) {
  super ("View");
}
parentFrame = parent;

treeViewMenu.setOpened (true);
toolTipsMenu.setOpened (true);
selectionModeMenu.setOpened (false);

manager = ToolTipManager.sharedInstance ();
manager.setEnabled (true);
manager.setInitialDelay (400);

add (colorMenu); //fontMenu)
addSeparator ();
add (treeViewMenu);
add (toolTipsMenu);
add (selectionModeMenu);
addSeparator ();
add (refreshMenu);

/**
 * These are not implemented yet
 * Take these lines out when they are implemented
 */
//colorMenu.setDisabled (false);
//fontMenu.setDisabled (false);

colorMenu.addCommandActionCommand ("Changes the current color of the
editor");
fontMenu.addCommandActionCommand ("Changes the current font of the
editor");
treeViewMenu.addCommandActionCommand ("Makes visible/hides the tree
view");
toolTipsMenu.addCommandActionCommand ("Enables/Disables tool tips");
selectionModeMenu.addCommandActionCommand ("Default displays the
draw area after each insertion of a component" +
"on the drawing area");

colorMenu.addMouseLisenter (parentFrame.getStatusBar ());
fontMenu.addMouseLisenter (parentFrame.getStatusBar ());
treeViewMenu.addMouseLisenter (parentFrame.getStatusBar ());
toolTipsMenu.addMouseLisenter (parentFrame.getStatusBar ());
refreshMenu.addMouseLisenter (parentFrame.getStatusBar ());
selectionModeMenu.addMouseLisenter (parentFrame.getStatusBar ());

selectionModeMenu.addCommandActionListener (this);

/**
 * Handles the menu events that occur when one of the menu items
 * is selected
 * @param The associated ActionEvent
 */
public void actionPerformed (ActionEvent e)
{
    if (e.getSource () == colorMenu)
    {
        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;
                    break;
                }
            }
            parentFrame.getDrawPanel ().setCurrentColor (colorIndex);
        }
    }
    else if (e.getSource () == fontMenu)
    {
        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;
                    break;
                }
            }
            parentFrame.getDrawPanel ().setCurrentFont (fontIndex);
        }
    }
    else if (e.getSource () == treeViewMenu)
    {
        if (!treeViewMenu.isOpened ()
        parentFrame.getSplitPane ().setDividerLocation (0.0);
        else
        parentFrame.getSplitPane ().setDividerLocation
        (parentFrame.getSplitPane ().getDividerLocation () +
        (parentFrame.getSplitPane ().getDividerLocation ());
    }
    else if (e.getSource () == toolTipsMenu)
    {
        if (toolTipsMenu.isOpened ()
        parentFrame.getSplitPane ().setDividerLocation (0.0);
        else
        parentFrame.getSplitPane ().setDividerLocation
        (parentFrame.getSplitPane ().getDividerLocation ());
    }
package caps.GraphEditor;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import capts.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));
    }

    // End of the class GE_ViewMenu

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 (GrammarCheckCheck.isNotValid (newId, GrammarCheck.ID)) {
                model.addElement (newId);
                inputArea.setSelectedValue (newId, true);
            }
        }
    }
    else if (e.getSource () == deleteButton) {
        int index = inputArea.getSelectedIndex ();
        if (index >= 0) // If there is a selected elements
        {
            model.removeElementAt (index);
        }
    }
    else if (e.getSource () == editButton) {
        int index = inputArea.getSelectedIndex ();
        if (index >= 0) {
            String editedId = showInputDialog ((String) inputArea.getSelectedValue ());
            if (editedId != null & editedId.length () != 0) {
                if (GrammarCheckCheck.isNotValid (editedId, GrammarCheck.ID)) {
                    model.removeElementAt (editedId, index);
                }
            }
        }
    }
    else if (editedId.length () == 0)
        model.removeElementAt (index);
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 ()
public JMenuItem getDeleteMenuItem ()
{
    return deleteMenuItem;
}

public JMenuItem getPropMenuItem ()
{
    return propMenuItem;
}

public void showPopupMenu (boolean isEdge, int x, int y)
{
    if (isEdge) {
        decomposeMenuSetEnabled (false);
        colorMenuSetEnabled (false);
    }
    else {
        decomposeMenuSetEnabled (true);
        colorMenuSetEnabled (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.Powl.*;
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) {
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
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 ("Save required");
    }

    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 mouseReleased (MouseEvent e)
    {
    }

} // End of the class StatusBar

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);
promptLabel = new JLabel();
dialog.getContentPane().add(promptLabel, BorderLayout.NORTH);

inputArea = new JTextArea();
inputArea.setLineWrap (true);
inputArea.setBorder (BorderFactory.createRaisedBevelBorder ());
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 ();
                else { showErrorDialog("Illegal value entered");
                    errorStatus = true;
                }
            }
            else (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);
}
package caps.GraphEditor;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.Border;
import caps.Pddl.DataTypes;
import caps.Pddl.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 DURANLI'OGLU
 * @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);
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);
        PsiBuilder.setCurrentOp (parent);
        PsiBuilder.ReInit (new StringReader (TextEditor.getString ()));
    try {
        PsiBuilder.operator_spec (false);
    } catch (ParseException ex) {
        /* This is already caught in GrammarCheck */
    }
}
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.

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 DURANLI.IOGLU
 * @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);
```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)

}  // End of the class TreePanel.
```
package caps.GraphEditor;
import javax.swing ImageIcon;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.TreeCellRenderer;
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
         */
    }
}
```java
package caps.GraphEditor;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import caps.Pddl.*;
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 nameLabel;
    JPanel triggerPanel;
    JPanel timingPanel;
    JPanel guardPanel;
    JPanel keywordsPanel;
    JPanel controlPanel;

    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);
        g.fillRect(0, 0, getWidth() - 1, getHeight() - 1);
    }
```
JButton metRegByButton;
JButton periodRegByButton;
JButton fwRegByButton;
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;
    setSize (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 ());
    nameField = new JTextField (10);
    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.griddx++;
    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);
    ifConditionField = new TextField (10, 20, TextArea.SCROLLBARS_VERTICAL_ONLY);
    ifConditionField.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.gridx = 1; gbc.gridy = 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.gridxdx++;
    triggerPanel.add (ifConditionField, gbc);
    gbc.gridwidth = 1; gbc.griddx = 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 (10);
    metUnitsCombo = getUnitsComboBox ();
    metReqByButton = new JButton("Required By");
    metReqByButton.addActionListener (this);
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(" Finish Within ");
gbc.gridx = 1; gbc.gridy = 0;
timingPanel.add (timingCombo, gbc);
gbc.gridx = 1; gbc.gridy = 1;
timingPanel.add (metLabel, gbc);
gbc.gridx = 1; gbc.gridy = 1;
timingPanel.add (periodLabel, gbc);
gbc.gridx = 1; gbc.gridy = 2;
timingPanel.add (periodCombo, gbc);
gbc.gridx = 1; gbc.gridy = 2;
timingPanel.add (periodReqByButton, gbc);
gbc.gridx = 1; gbc.gridy = 3;
timingPanel.add (periodField, gbc);
gbc.gridx = 1; gbc.gridy = 3;
timingPanel.add (periodUnitsCombo, gbc);
gbc.gridx = 1; gbc.gridy = 3;
timingPanel.add (periodReqByButton, gbc);
gbc.gridx = 1; gbc.gridy = 4;
timingPanel.add (cancelButton, gbc);
gbc.gridx = 1; gbc.gridy = 7;
timingPanel.add (okButton, gbc);
gbc.gridx = 1; gbc.gridy = 2;
timingPanel.add (cancelButton, gbc);
gbc.gridx = 1; gbc.gridy = 3;
timingPanel.add (okButton, gbc);
gbc.insets = new Insets(1, 15, 1, 15);
guardsPanel = new JPanel(new GridBagLayout());
guardsPanel.setBorder (BorderFactory.createTitledBorder (" GUARDS "));
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);
guardsPanel.add (exceptionGuardsButton, gbc);
guardsPanel.add (exceptionListButton, gbc);
guardsPanel.add (timerOpsButton, gbc);
gbc.gridx = 0; gbc.gridy = 1;
guardsPanel.add (exceptionListButton, gbc);
guardsPanel.add (timerOpsButton, gbc);
guardsPanel.add (exceptionGuardsButton, gbc);
guardsPanel.add (exceptionListButton, gbc);
guardsPanel.add (timerOpsButton, gbc);
guardsPanel.add (okButton, gbc);
guardsPanel.add (cancelButton, gbc);
keywordsPanel = new JPanel (new FlowLayout ());
keywordsPanel.setBorder (BorderFactory.createTitledBorder (" Keywords "));
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);
okPanel.add (okButton, gbc);
okPanel.add (cancelButton, gbc);
okPanel.add (helpButton, gbc);
namePanel = new JPanel();
box = new Box.createVerticalStrut (5);
box = new Box.createHorizontalStrut (5);
box = new Box.createVerticalStrut (5);
box = new Box.createHorizontalStrut (5);
box = new Box.createVerticalStrut (3);
getContentPane ().add (box, BorderLayout.CENTER);
public JComboBox getUnitsCombo ()
{
    JComboBox c = new JComboBox ();
c.additem ("microsec");
c.additem ("ms");
c.additem ("sec");
c.additem ("min");
c.additem ("hours");
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 action performed
    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.getValue ());
        metUnitsCombo.setSelectedIndex (met.getUnits ());
    }
    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.getValue ())); periodUnitsCombo.setSelectedIndex (period.getUnits ());
        }
        else {
            periodField.setText (""");
            periodUnitsCombo.setSelectedIndex (1);
        }
        if (fw != null) {
            fwField.setText (String.valueOf (fw.getValue ()));
            fwUnitsCombo.setSelectedIndex (fw.getUnits ());
        }
        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.getValue ()));
        periodUnitsCombo.setSelectedIndex (mcp.getUnits ());
    }
    else {
        periodField.setText (""");
        periodUnitsCombo.setSelectedIndex (1);
    }
    if (mrt != null) {
        fwField.setText (String.valueOf (mrt.getValue ()));
        fwUnitsCombo.setSelectedIndex (mrt.getUnits ());
    }
    else {
        fwField.setText (""");
        fwUnitsCombo.setSelectedIndex (1);
    }
}

else {
    ifCondField.setText (v.getIfCondition ());
    if (tempVertex.isTerminator ()) {
        metLabel.setEnabled (false);
        metField.setEnabled (false);
        metUnitsCombo.setEnabled (false);
    }
}

public void setDisplayVertex (DisplayVertex v)
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);
    periodRegByButton.setEnabled(true);
    finishWithinLabel.setEnabled(true);
    finishWithinField.setText("*");
    finishWithinField.setEnabled(true);
    triggerRegByButton.setEnabled(true);
    triggerRegByButton.setEnabled(true);
    if (tempVertex.isTerminator()) {
        periodLabel.setEnabled(false);
        periodField.setEnabled(false);
        periodUnitsCombo.setEnabled(false);
        periodRegByButton.setEnabled(false);
        finishWithinLabel.setEnabled(false);
        finishWithinField.setEnabled(false);
        triggerRegByButton.setEnabled(false);
    }
}

setButtonText(triggerRegByButton, tempVertex.getTriggerRegmts());
setButtonText(triggerStreamsButton, tempVertex.getTriggerStreamsList());
setButtonText(metRegByButton, tempVertex.getMetRegmts());
setButtonText(periodRegByButton, tempVertex.getPeriodRegmts());
setButtonText(fwRegByButton, tempVertex.getFinishWithinRegmts());
setButtonText(fwRegByButton, tempVertex.getMrcRegmts());
setButtonText(outputGuardsubmitButton, tempVertex.getOutputGuardList());
setButtonText(exceptionGuardsubmitButton, tempVertex.getExceptionGuardList());
setButtonText(exceptionListButton, tempVertex.getExceptionList());
setButtonText(timerOpsButton, tempVertex.getTimerOpList());

}
else if (e.getSource () == outputGuardsButton) {
    TextEditor.openDialog ("Operator Output Guard", "View or Edit Operator Output Guard Equation");
    tempVertex.setOutputGuardList (1, GrammarCheck.CHECK_OPERATOR_GUARDS, true);
    tempVertex.setOutputGuardList (TextEditor.getString () );
    setButtonText (outputGuardsButton, TextEditor.getText () );
}
else if (e.getSource () == exceptionGuardsButton) {
    TextEditor.openDialog ("Operator Exceptions", "View or Edit Operator Exceptions");
    tempVertex.getExceptionGuardList ();
    tempVertex.setExceptionGuardList (TextEditor.getString () );
    setButtonText (exceptionGuardsButton, TextEditor.getText () );
}
else if (e.getSource () == exceptionListButton) {
    TextEditor.openDialog ("Operator Exceptions", "View or Edit Operator Exceptions");
    tempVertex.getExceptionList ();
    tempVertex.setExceptionList (TextEditor.getString () );
    setButtonText (exceptionListButton, TextEditor.getText () );
}
else if (e.getSource () == timerOpsButton) {
    TextEditor.openDialog ("Operator Timers", "View or Edit Operator Timers");
    tempVertex.getTimerOpList ();
    tempVertex.setTimerOpList (TextEditor.getString () );
    setButtonText (timerOpsButton, TextEditor.getText () );
}
else if (e.getSource () == keywordsButton) {
    IDListEditor.openDialog (tempVertex.getKeywordList ());
    tempVertex.setKeywordList (IDListEditor.getIdList () );
    setButtonText (keywordsButton, IDListEditor.getText () );
}
else if (e.getSource () == informalDescButton) {
    TextEditor.openDialog ("Informal Design Description", "View or Edit Informal Design Description");
    tempVertex.getInformalDesc ();
    tempVertex.setInformalDesc (TextEditor.getString () );
    setButtonText (informalDescButton, TextEditor.getText () );
}
else if (e.getSource () == formalDescButton) {
    TextEditor.openDialog ("Formal Design Description", "View or Edit Formal Description");
    tempVertex.getFormalDesc ();
    tempVertex.setFormalDesc (TextEditor.getString () );
    setButtonText (formalDescButton, TextEditor.getText () );
}
else if (e.getSource () == okButton) {
    boolean exceptionOccurred = false;
    String str = fieldName.getText ();
    if (!GrammarCheck.isValid (str, GrammarCheck.ID)) {
        showErrorDialog ("Illegal vertex name");
        exceptionOccurred = true;
    }
    if (timingCombo.getSelectedIndex () != Vertex.NON_TIME_CRITICAL) {
        str = metaField.getText ();
        if (!targetVertex.isTerminator ()) {
            if (!GrammarCheck.isValid (str, GrammarCheck.INTEGER_LITERAL)) {
                showErrorDialog ("Illegal value for meta 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");
                }
            }
        }
        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");
                }
            }
        }
    }
    if (!exceptionOccurred) {
        targetVertex.setLabel (fieldName.getText ());
        targetVertex.setImpLanguage ((String) languageCombo.getSelectedItem ());
        targetVertex.setTriggerType (triggerCombo.getSelectedIndex ());
        if (triggerCombo.getSelectedIndex () != Vertex.UNPROTECTED)
targetVertex.setTriggerStreamsList
(tempVertex.getTriggerStreamsList ());
else
    targetVertex.setTriggerStreamsList (new Vector ());
if (timingCombo.getSelectedIndex () != Vertex.NON_TIME_CRITICAL &
    !(targetVertex.isTerminator ())) {
    targetVertex.setMet (new PSDLTime (Integer.parseInt
    (metField.getText ())),
    metUnitsCombo.getSelectedIndex
    ());
    targetVertex.setMetRegmts (tempVertex.getMetRegmts ());
} else if ( !(targetVertex.isTerminator ()) ) {
    targetVertex.setMet (null);
    targetVertex.setMetRegmts ().removeAllElements ();
}
if (timingCombo.getSelectedIndex () == Vertex.PERIODIC) {
    if (periodField.getText ().length () != 0) {
        targetVertex.setPeriod (new PSDLTime (Integer.parseInt
        (periodField.getText ())),
        periodUnitsCombo.getSelectedIndex ());
        targetVertex.setPeriodRegmts (tempVertex.getPeriodRegmts
        ());
    } else {
        targetVertex.setPeriod (null);
        targetVertex.setPeriodRegmts ().removeAllElements ();
    }
    if (fwField.getText ().length () != 0) {
        targetVertex.setFinishWithin (new PSDLTime (Integer.parseInt
        (fwField.getText ())),
        fwUnitsCombo.getSelectedIndex ());
        targetVertex.setFinishWithinRegmts (tempVertex.getFinishWithinRegmts
        ());
    } else {
        targetVertex.setFinishWithin (null);
        targetVertex.setFinishWithinRegmts ().removeAllElements
        ();
    }
    targetVertex.setMcpRegmts ().removeAllElements ();
    targetVertex.setMrtRegmts ().removeAllElements ();
} else if (timingCombo.getSelectedIndex () == Vertex.SPORADIC) {
    if (periodField.getText ().length () != 0) {
        targetVertex.setPeriod (new PSDLTime (Integer.parseInt
        (periodField.getText ())),
        periodUnitsCombo.getSelectedIndex ());
        targetVertex.setPeriodRegmts (tempVertex.getPeriodRegmts
        ());
    } else {
        targetVertex.setPeriod (null);
        targetVertex.setPeriodRegmts ().removeAllElements ();
    }
    if (fwField.getText ().length () != 0) {
        targetVertex.setFinishWithin (new PSDLTime (Integer.parseInt
        (fwField.getText ())),
        fwUnitsCombo.getSelectedIndex ());
        targetVertex.setFinishWithinRegmts (tempVertex.getFinishWithinRegmts
        ());
    } else {
        targetVertex.setFinishWithin (null);
        targetVertex.setFinishWithinRegmts ().removeAllElements
        ();
    }
    targetVertex.setMcpRegmts ().removeAllElements ();
    targetVertex.setMrtRegmts ().removeAllElements ();
} else if (changeStatus == TO_OPERATOR) {
    targetVertex.setTerminator (false);
    parentFrame.getDrawPanel ().changeLevel ((Vertex)
    parentFrame.getDrawPanel ().getParent ());
} else if (changeStatus == TO_TERMINATOR) {
DataFlowComponent dfc;
for (Enumeration enum =
    targetVertex.breadthFirstEnumeration (); enum.hasMoreElements ();)
    { 
        dcf = (DataFlowComponent) enum.nextElement ();
        if (dfc instanceof Vertex) {
            ((Vertex) dcf).setTerminator (true);
            ((Vertex) dcf).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.setSizeRequired (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
        // Restore ro older one
        ) 
        targetVertex.isTerminator () && changeStatus == TO_OPERATOR
        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.getMetRegts ().removeAllElements ();
    tempVertex.getPeriodRegts ().removeAllElements ();
    tempVertex.getFinishWithinRegts ().removeAllElements ();
    tempVertex.getMrtRegts ().removeAllElements ();
    tempVertex.getMcpRegts ().removeAllElements ();
    if (tempVertex.getUpdatedRegts () == 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 ("..."))
            b.setText (b.getText ().trim () + " ...");
        }
}
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;
/**
 * The met of a Vertex or the latency of a Stream.
 * 
 * protected FSDLTime 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 met1Font;  // 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);
    met1Font = 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;
}
/** * 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 get1Font ()
{
    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;
}

/** * Sets the x-component of the offset of the met (or latency).
 */
public int setMetXOffset ()
{
    return metXOffset;
}

/** * Returns the x-component of the offset of the met (or latency).
 */
public int getLabelXOffset ()
{
    return labelXOffset;
}

/** * 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);
    labelFont = 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);
    metFont = 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
package caps.Pddl;

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;
                break;
            }
        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 reaing 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;
        int tokType;
        int counter = 0;
        try {
            tok.wordChars (33, 126);
            tok.eolIsSignificant (true);
            while (tokType = tok.nextToken () != StreamTokenizer.TT_ROF) {
                if (tokType == StreamTokenizer.TT_WORD &
                    tok.sval.equalsIgnoreCase ("TYPE") {
                    tempStr = getNextToken (tok);
                    str = tempStr;
                    types.addElement (str);
                    do {
                        tempStr = getNextToken (tok);
                    } while (!tempStr.equals ("end") ||
                }
            }
        } catch (IOException ex) {
            System.out.println (ex);
        }
    }
}

// This will be called when reaing 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;
    int tokType;
    int counter = 0;
    try {
        tok.wordChars (33, 126);
        tok.eolIsSignificant (true);
        while (tokType = tok.nextToken () != StreamTokenizer.TT_ROF) {
            if (tokType == StreamTokenizer.TT_WORD &
                tok.sval.equalsIgnoreCase ("TYPE") {
            tempStr = getNextToken (tok);
            str = tempStr;
            types.addElement (str);
            do {
                tempStr = getNextToken (tok);
            } while (!tempStr.equals ("end") ||
        }
    } catch (IOException ex) {
        System.out.println (ex);
    }
}
```java
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.
```

package caps.Fsd1;

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 DURANLOGLU
 * @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 () ;
    }

    /**
     * The constructor for this class.
     */
    */
    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 () ;
    }

    /**
     * The constructor for this class.
     */
    */
    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 () ;
    }

    /**
     * The constructor for this class.
     */
    */
    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 () ;
    }
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;
    }
    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;
}
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 ****
/**
 * 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 control point.
 * @param y the y component of the new control 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));
}

/**
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
    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 ();
}
package caps.Psd1;
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
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 milliseconds.
 */
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 setValue(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 setUnits(int timeUnits) {
    units = timeUnits;
}

/**
 * Sets the time unit to the specified argument.
 */
public void setUnits(String u)
{
    if (u == "microsec")
        units = microsec;
    else if (u == "ms")
        units = ms;
    else if (u == "sec")
        units = sec;
}
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 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

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 initial radius of a Vertex.
     */
    public static final int INITIAL_RADIUS = 35;

    /**
     * The constant value for non-time critical 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;

    /**
     * The constant value for Vertices that have "BY Some" triggering
     * construct.
     */
    public static final int BY_SOME = 1;
protected Vector mrtReqsmts;
protected int timingType;
protected int triggerType;
protected Vector triggerReqsmts;
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 specReqsmts;

/**
 * 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++;
    id = ++UNIQUE_ID;
}
period = null;
finishWithin = null;
mcp = null;
mrt = null;
metReqs = new Vector (0, 2);
periodReqs = new Vector (0, 2);
finishWithinReqs = new Vector (0, 2);
mcpReqs = new Vector (0, 2);
mrtReqs = new Vector (0, 2);
triggerReqs = new Vector (0, 2);
triggerStreamList = new Vector (0, 2);
ifCondition = "";
outputGuardList = "";
exceptionGuardList = "";
exceptionList = "";
timerOpList = "";
keywordList = "";
informalDesc = "";
formalDesc = "";
timerList = new Vector (0, 2);
graphDesc = "";
genericList = "";
specReqs = new Vector (0, 2);
specReqs.addElement (new Vector (0, 2));
specReqs.addElement (new Vector (0, 2));
specReqs.addElement (new Vector (0, 2));

implLanguage = "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;
}
/**
 * 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;
}

/**
 * Returns the width of this component.
 * @return the width of this component.
 */
public int getWidth ()
{   
    return 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;
}
* 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) specReqsmts.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) specReqsmts.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) specReqsmts.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) specReqsmts.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 SDLTime getPeriod ()
{
  return period;
}
*/
public void setMrt (PSDLTime mr)
{
    mrt = mr;
}

/**
 * Returns the implementation language of this Vertex.
 */
public String getImplLanguage ()
{
    return impLanguage;
}

/**
 * Sets the implementation language to the specified value.
 */
public void setImplLanguage (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;
}
public Vector getTriggerReqsmts ()
{
    return triggerReqsmts;
}

/**
 * Sets the trigger requirements to the specified value.
 */
public void setTriggerReqsmts (Vector v)
{
    triggerReqsmts = 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;
** Sets the output guard list to the specified value.

```java
public void setOutputGuardList (String s)
{
    outputGuardList = s;
}
```

** Returns the exception guard list of this Vertex.

```java
public String getExceptionGuardList ()
{
    return exceptionGuardList;
}
```

** Sets the exception guards list to the specified value.

```java
public void setExceptionGuardList (String s)
{
    exceptionGuardList = s;
}
```

** Returns the exception list of this Vertex.

```java
public String getExceptionList ()
{
    return exceptionList;
}
```

** Sets the exception list to the specified value.

```java
public void setExceptionList (String s)
{
    exceptionList = s;
}
```

** Returns the timer op list of this Vertex.

```java
public String getTimerOpList ()
{
    return timerOpList;
}
```

** Sets the timer op list to the specified value.

```java
public void setTimerOpList (String s)
{
    timerOpList = s;
}
```

** Returns the informal description of this Vertex.

```java
public String getInformalDesc ()
{
    return informalDesc;
}
```

** Sets the informal description to the specified value.

```java
public void setInformalDesc (String s)
{
    informalDesc = s;
}
```

** Returns the formal description of this Vertex.

```java
public String getFormalDesc ()
{
    return formalDesc;
}
```

** Sets the formal description to the specified value.

```java
public void setFormalDesc (String s)
{
    formalDesc = s;
}
```

** Returns the keywords of this Vertex.

```java
public Vector getKeywordList ()
{
    return keywordList;
}
```

** Sets the keywords to the specified value.

```java
public void setKeywordList (Vector v)
{
    keywordList = v;
}
/**
 * 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 getSpecRegmts ()
{
    return specRegmts;
}

/**
 * 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)
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 " + ".\n");
        spec = spec.concat (" SPECIFICATION\n");
        if (genericList.length () != 0)
            spec = spec.concat (genericList + "\n");
        en = ((Vector) specRegExs.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) specRegExs.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) specRegExs.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 () + " INITIALY " + ((Edge) d).getInitialValue () + "\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 (!metRegExs.isEmpty ()
                spec = spec.concat (" REQUIRED BY " + extractList
                (metRegExs) + "\n");
            }
            if (keywordList.size () > 0)
                spec = spec.concat (" KEYWORDS " + extractList (keywordList
                ) + "\n");
            }
            spec = spec.concat (
            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;
        }
/**
 * Called from getSpecification.
 * Extracts the string parameter and reformat 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 ()
{

writer.write(""
" + v.getY () + "\n");
PROPERTY radius = " + (v.getWidth () / 2) + "\n";
PROPERTY color = " + v.getColor () + "\n";
PROPERTY label_font = " +
PROPERTY label_x_offset = " +
PROPERTY label_y_offset = " +
PROPERTY met_font = " + v.getMetFontIndex
PROPERTY met_unit = " +
PROPERTY met_x_offset = " +
PROPERTY met_y_offset = " +
PROPERTY is_terminator = " +

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 () + "\n");
else
writer.write (e.getSource ().getLabel () + "+ + e.getSource
().getId () + " +
(e.getSource ().getId () - 1));
writer.write("\n");
if (e.getDestination () instanceof External)
writer.write (e.getDestination ().getLabel () + "\n");
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";

e.getLabelFontIndex () + "\n";
PROPERTY label_x_offset = " +
PROPERTY label_y_offset = " +
PROPERTY latency_font = " +
PROPERTY latency_unit = " +
(e.getMet () == null) ? 1 : e.getMet ().getTimeUnits
());
PROPERTY latency_x_offset = " +
PROPERTY latency_y_offset = " +
PROPERTY spline = "\n";
writer.write("\n");
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 () +
(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 ()");
)
else if (!e.isStateStream())
    str = str.concat(" + 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 () + " + 1 + " + v.getTriggerType () + " : ");
    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 ");
    writer.write (v.extractList (v.getTriggerStreamsList ()) + "\n");
    if (v.getTriggerRegmts ().size () != 0)
        writer.write (" + v.extractList (v.getTriggerRegmts ()));
}

public static void period (Vertex v)
{
    if (v.getPeriod () != null)
        writer.write (" + PERIOD " + v.getPeriod ().toString () + "\n");
    if (v.getPeriodRegmts ().size () != 0)
        writer.write (" + v.extractList (v.getPeriodRegmts ()) + "\n");
}

public static void finishWithin (Vertex v)
{
    if (v.getFinishWithin () != null)
        writer.write (" + FINISH WITHIN " + v.getFinishWithin ().toString () + "\n");
    if (v.getFinishWithinRegmts ().size () != 0)
        writer.write (" + v.extractList (v.getFinishWithinRegmts ()) + "\n");
}

public static void mcp (Vertex v)
{
    if (v.getMcp () != null)
        writer.write (" + MINSUM CALLING PERIOD " + v.getMcp ().toString () + "\n");
    if (v.getMcpRegmts ().size () != 0)
        writer.write (" + v.extractList (v.getMcpRegmts ()) + "\n");
}

public static void mrt (Vertex v)
{
    if (v.getMrt () != null)
        writer.write (" + MAXIMUM RESPONSE TIME " + v.getMrt ().toString () + "\n");
    if (v.getMrtRegmts ().size () != 0)
        writer.write (" + v.extractList (v.getMrtRegmts ()) + "\n");
}

public static void outputGuards (Vertex v)
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);
PddlParser.reInit (reader);
        boolean flag = true;
        try {
            switch (kind) {
            case ID :
                PddlParser.id ();
                break;
            case TYPE_NAME :
                PddlParser.type_name ();
                break;
            case INTEGER_LITERAL :
                PddlParser.integerLiteral ();
                break;
            case INITIAL_EXPRESSION :
                PddlParser.initialExpression ();
            }
break;
case EXPRESSION :
    PSDLParser.expression ();
    break;
    return flag; }
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.psd1 ();
    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;
    }
LIST OF REFERENCES


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

295
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