FINAL DESIGN REPORT

BELLATRIX

DIGITAL CIRCUIT SIMULATOR

Emin ÖZCAN - 1298090
Mehtap Ayfer PARLAK - 1347855
Mehmet Ergin SEYFE - 1298215
Ilgın YARIMAĞAN - 1409101
Eren YILMAZ – 1298470

Table of Contents

1 Introduction

1.1 Purpose of the System

Designed for basically educational purposes, Bellatrix is capable of performing the simulation of digital circuits consisting of various components and wires. Users will be able to add custom defined components as well as well-known components such as multiplexers, flip flop, and gates etc...

Bellatrix enables extended functionalities such as compatibility with Diglog file format, option to print the circuit schema as a PDF file which are basically aimed to ease the jobs of students that are taking logic design laboratories. Users will be able to save a Bellatrix file in several formats such as JPEG GIF, PNG, PS, PDF, HTML.

One of the primary features that distinguishes Bellatrix from other digital circuit simulators is the powerful script support it offers to its users. Scripting will mainly provide users a way to test the circuit more effectively. Users will either enter the script commands to the script console or execute a script file. Scope of scripting also includes most of the capabilities provided by the GUI as well as testing. For example users will be able to add a component in a specified coordinate by scripting.

1.2 Design Goals

Extensibility:
The application should be able to accommodate additional functionality. In particular, our system is designed so that it can be extended to accept user defined gates. This property will also ease the implementation of basic components since they can be defined using the program. The GUI is also designed so that its features can be expanded. (ex: the Edit menu).

Robustness:
The system should be able to manage invalid user inputs or inconsistent conditions. It provides error checking to ensure the right input format and returns errors and warnings to the user.

Reliability:
The system should produce the expected output for a valid input at all times.

Functionality:
The system should function according to the requirements specified in Requirements Analysis Report.

Usability:
The GUI should be user friendly. The goal is to provide the user an easy- to- use interface. The design of the GUI is based on that of Java based applications. This design is chosen due to the familiarity of most users with this kind of interface. It consists of a menu bar, which is further decomposed into sub menus. Text boxes, scrollbars and pop-up menus are used to enhance user/system interaction. The user is placed in a familiar environment, which eases the general use of the application.

1.3 Document Overview

This document explains the design of our application in detail and provides an overview of our program's functionality and implementation. Throughout this document the following major sections will be stated: UML Diagrams and Dynamic View. Prior to these sections, the Data Flow Diagrams are given again. In the UML Diagrams sections, the reader can find Use-Cases, Class Diagrams and Sequential Diagrams. Some of these diagrams are supported with descriptions. In the Dynamic View section, the actions that the user can do are explained.

2 Data Flow

2.1 Level 0 Data Flow Diagram

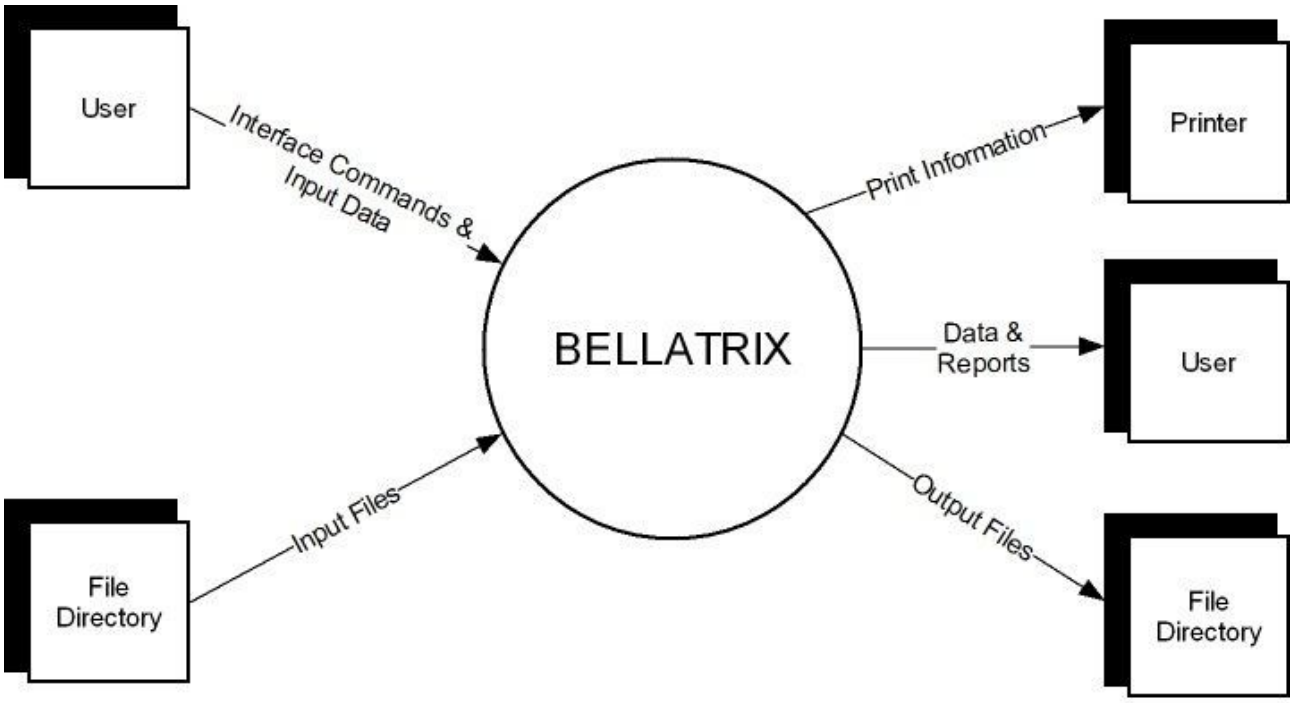


Figure 1: Level-0 DFD

2.2 Level 1 Data Flow Diagram

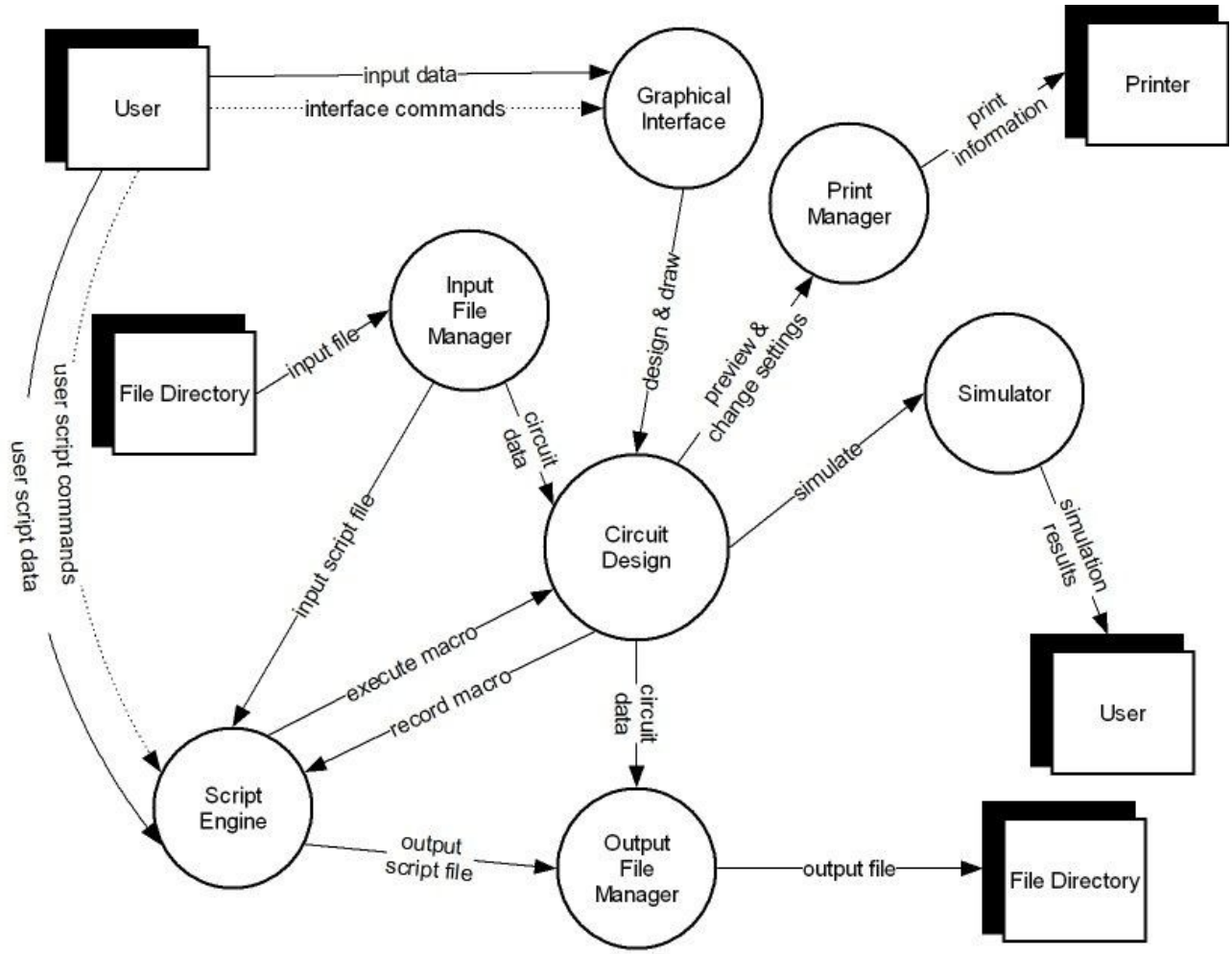


Figure 2: Level 1 DFD

2.3 Level 2 Data Flow Diagram – GUI

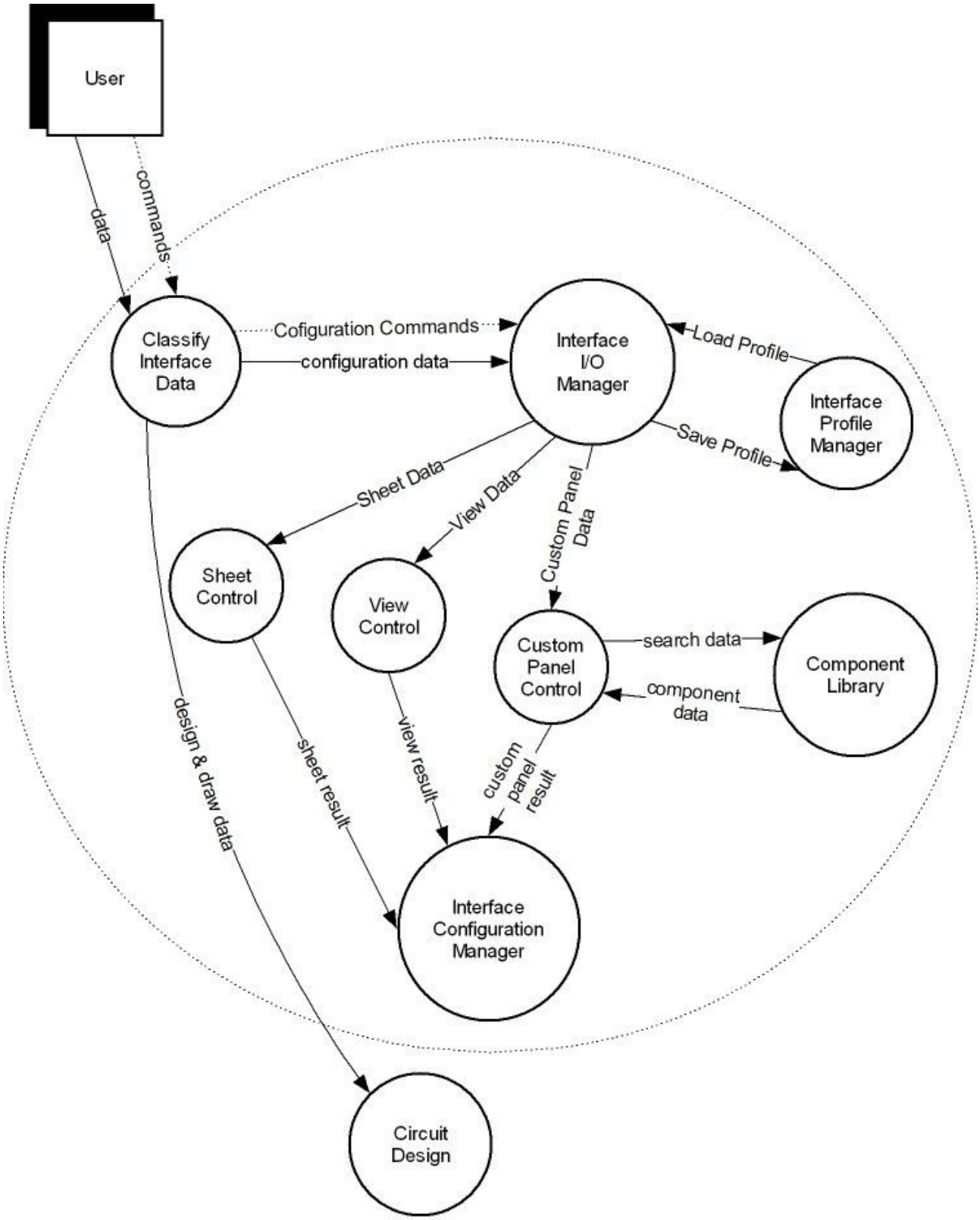


Figure 3: Level 2 DFD for GUI

2.4 Level 2 Data Flow Diagram – Scripting

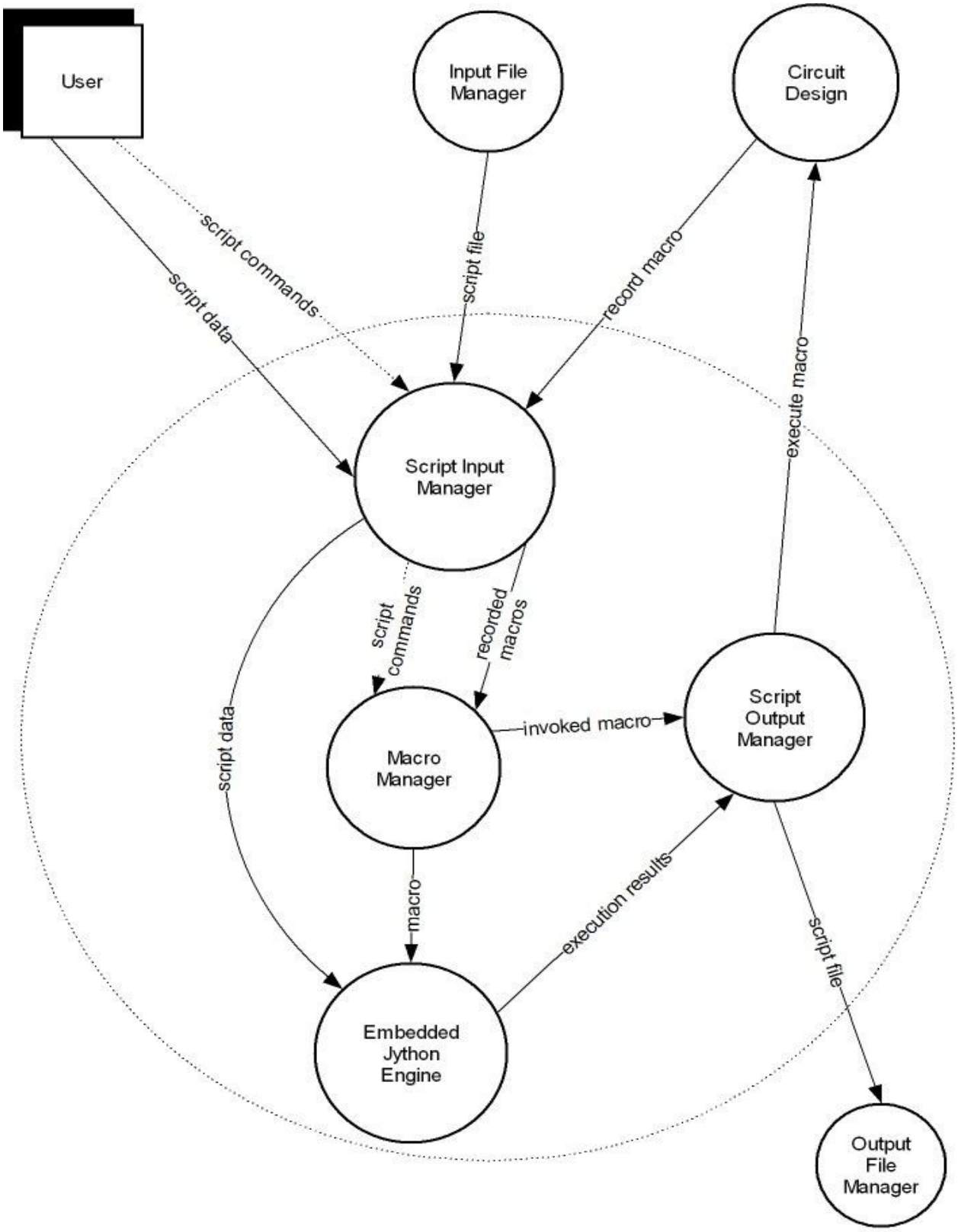


Figure 4: Level 2 DFD for Scripting

3 UML Diagrams

3.1 Use-Case

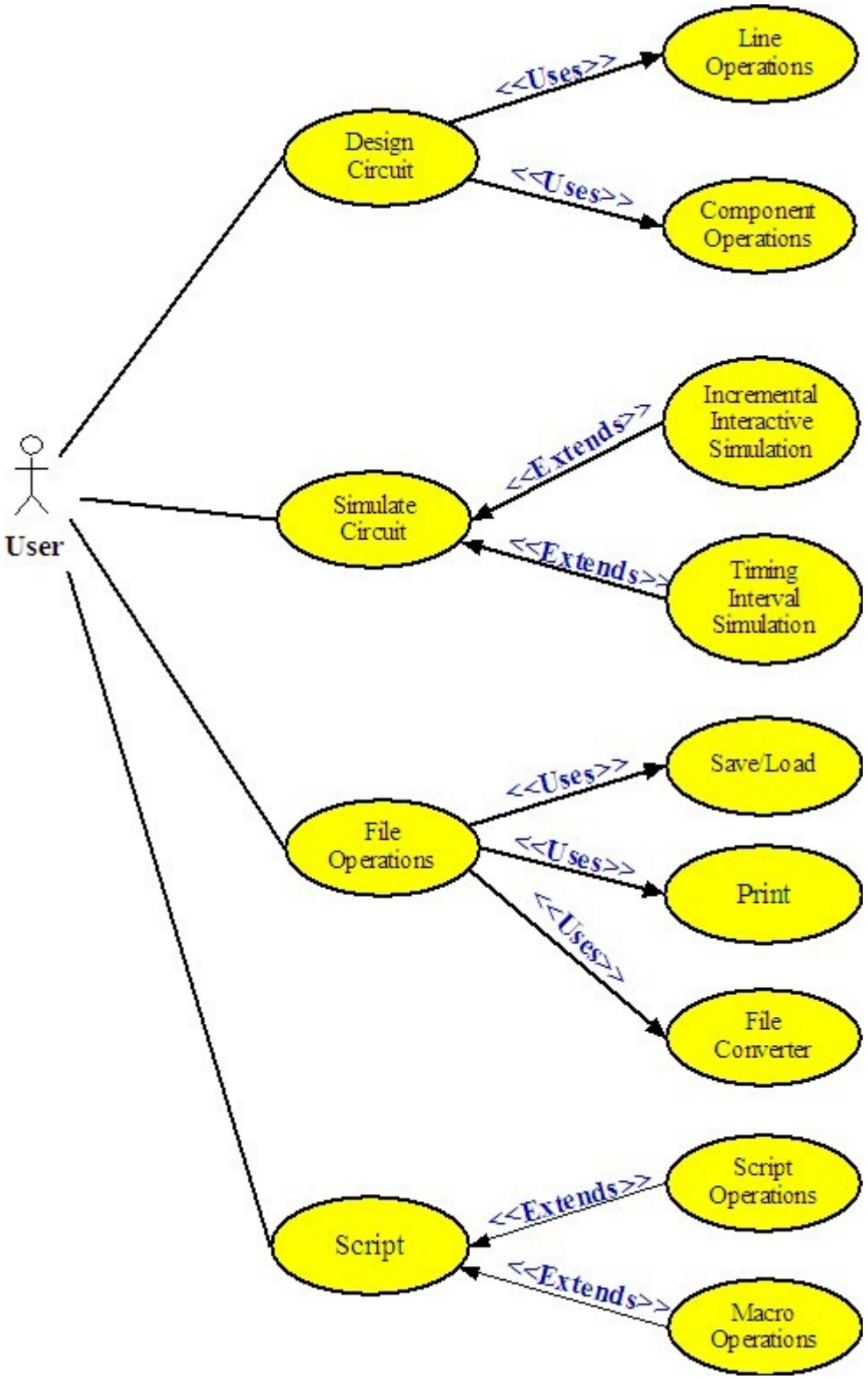


Figure 5: The General Use-Case Diagram of Bellatrix

Usage Scenario:

In Project Bellatrix, user can design a circuit by opening a project. He can save the project at any time he wants, and later he can open the same project from the disk. If he wants to open another project, he has to close the recent one.

In a project, there are three spaces. First of them is drawing area. Drawing area is composed of sheets. User can open numbers of sheets and can remove some of them from the drawing area if he wants. Also he can operate these sheets view. He can choose tile or cascade view for seeing the sheets more easily. In project, there is also a workspace area that is used for showing and searching the components. Any user can reach any basic or extended components from there. Also he can create his custom panel by searching the required components from the component library. In addition to these spaces, there is also a console which is used for giving external controller code.

User can design a circuit by using drawing area. He can choose any component from the workspace menu and drag it to the drawing area. At any time,he can move the component,rotate the component by right angles, and copy, cut, paste and delete the component at any time. User can select only one or lots of components and he can group them. There is also a redo, undo and clear support. For each component, user can see its properties and he can change its color and name. In addition to component drawing, user can draw connection lines. Firstly, he has to choose the line mode from the tool bar for activating the line drawing. After the activation of line drawing mode, he can draw wires. This activation prevents the accidental line drawing. Drawing area has default view settings. However, any user could want to change these settings. For this reason, system provides some competence to the user, such as changing the background color and controlling the grid view. Also there is a zooming support. After doing some changes on the view of drawing area, user can save them and load them any time he wants. Finally, user may want to deactivate the drawing area. So, there is hand mode support. In hand mode,user can only see the circuit ,namely, he can do nothing to the circuit.

In our program, user can test the circuit by using the simulator. Simulation mode is inactive when the program is in edit mode. If a user wants to simulate the circuit, he has to switch on simulation mode by selecting Run, from the tool bar or from Simulation on the menu bar. Also he can start the simulation by giving a script code from the console. Then he can pause, reset or stop the simulation, again by using the tool bar or console. In simulation of a circuit, a user can give inputs once or step by step.

Project Bellatrix supports the user with a macro peculiarity. User can record the drawing process by using the menu bar, or by giving an external code from the console. For recording the process, our system creates a script file. The user can load this file and execute the macro for watching it. In addition the automatic script file creation, our system gives a chance of manual script file creation to the user. User can open a script editor from Macro on the menu bar and he can write his script code here. After loading and executing his own script file, he will watch what this script does.

Finally, our system has a print support. After drawing a circuit, user can print out this circuit on a paper. If he wants to see the view of the paper before getting the print-out, he has to select the print preview button from the tool bar. User could want to change default print settings. Therefore, after clicking the print button on the tool bar, a new pop-up menu will be opened and user can change the settings,such as paper size or color.

3.2 Class Diagrams

3.2.1 General Class Diagram

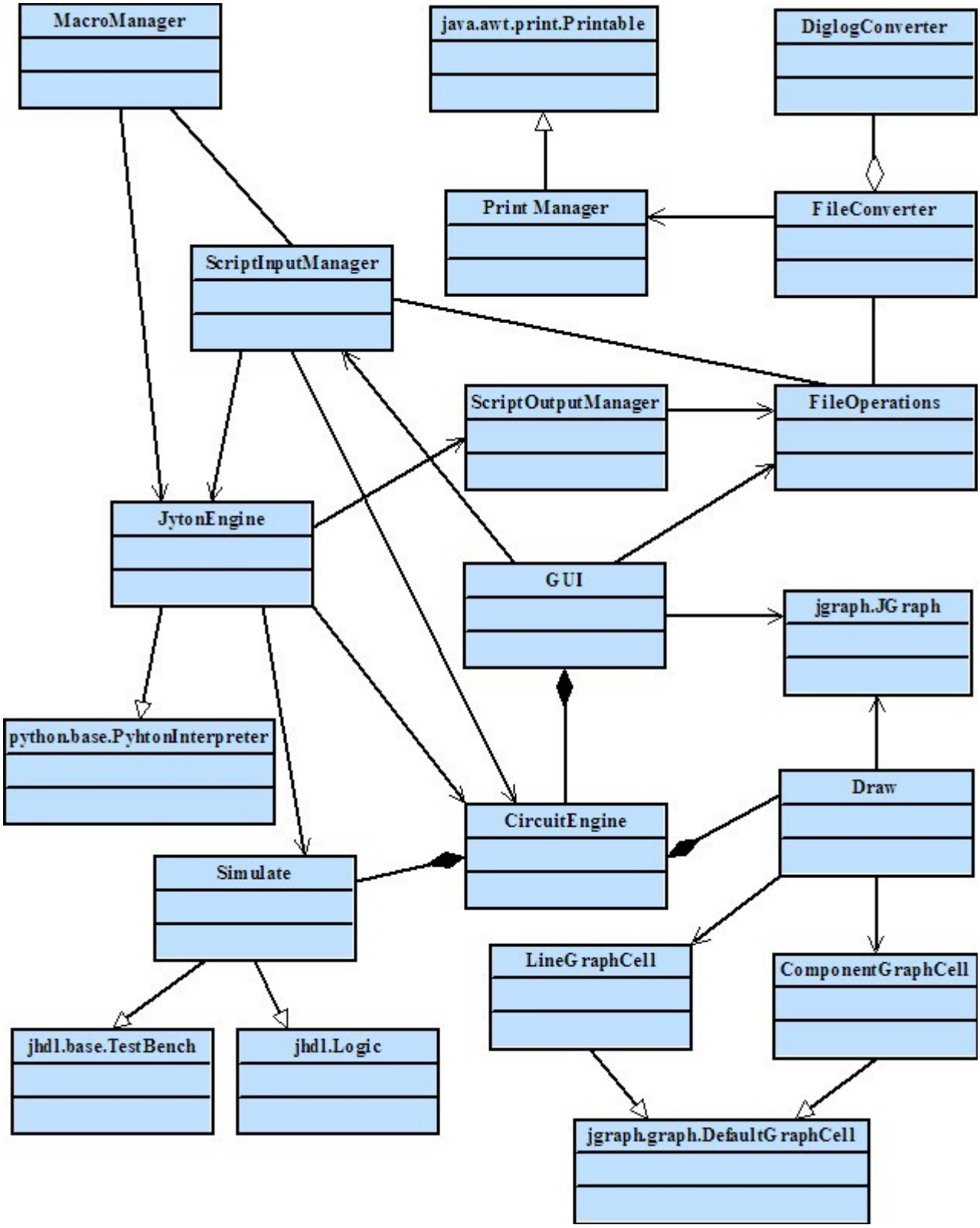


Figure 6: General Class Diagram of Bellatrix

3.2.2 Class Diagram : Drawing

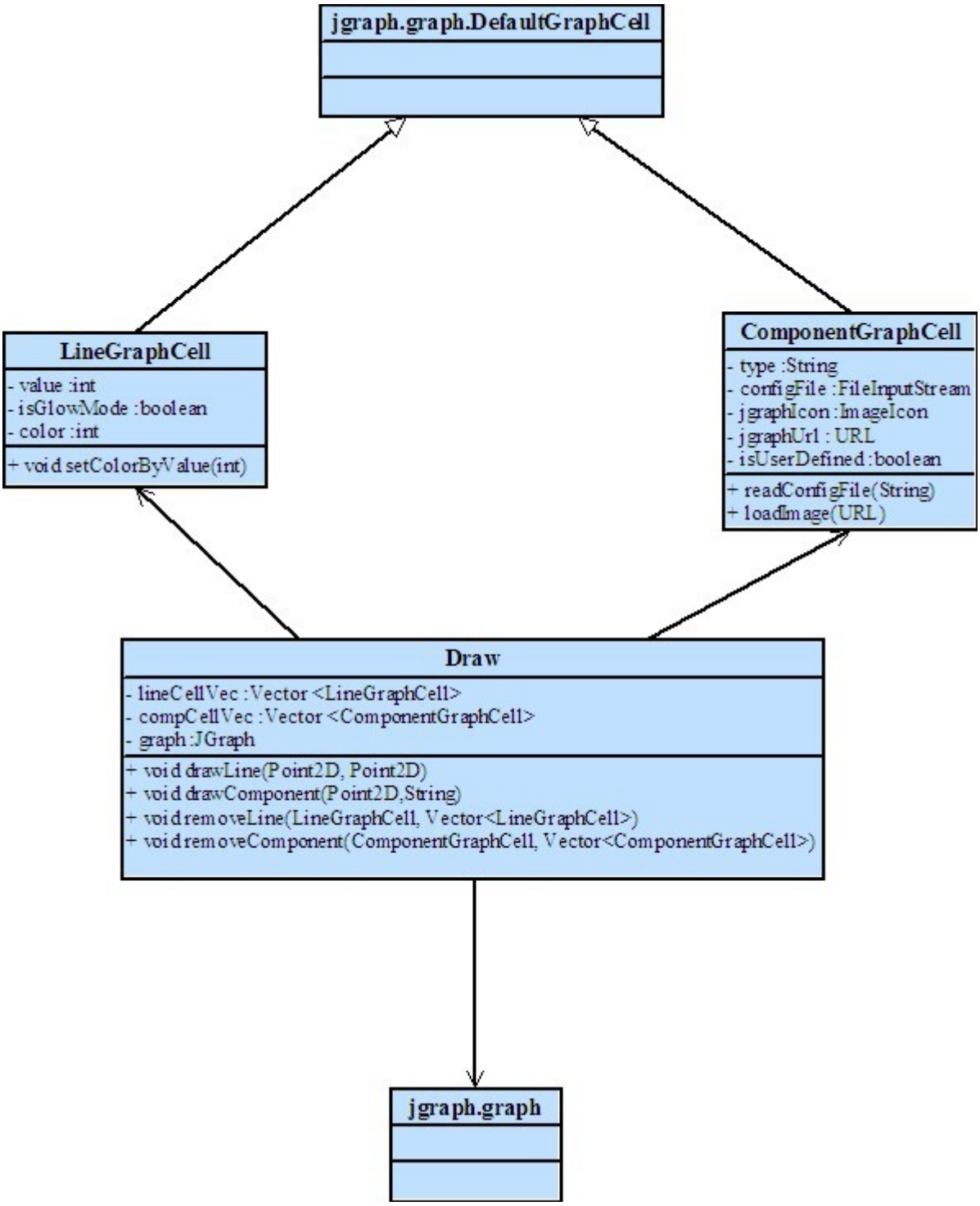


Figure 7: Class Diagram for Drawing

When the user drops a gate in the circuit drawing area of the program, or, when the user draws a line, a Draw class object will be instantiated by the CircuitEngine class. All of the drawing process will be done by the Draw class. Thanks to JGraph library, drawing is very easy. Let us continue with explaining how JGraph draw objects and how Draw class uses it and the subclasses of Draw, which are LineGraphCell and ComponentGraphCell.

First of all, JGraph library has a property that it displays its data by drawing individual elements. Each element displayed by the graph contains exactly one item of data, which is called a cell. A cell may either be a vertex(gate) or an edge(line). Vertices may have neighbors or not, and edges may have source and target vertices or not, depending on whether they are connected. Since a line or a gate has to be accessed from a cell, the line class and the component class extend from the DefaultGraphCell of JGraph library. DefaultGraphCell extends from the DefaultMutableTreeNode which is the general-purpose node in Jtree.

LineGraphCell and ComponentGraphCell classes represent for line and component, respectively, they inherit the DefaultGraphCell. LineGraphCell class has three private objects which are value, isGlowMode and color. Value and color are integer and isGlowMode is boolean. Color and isGlowMode fields are used in line drawing and the value field is used in simulation. Value is defined -1 as default,which shows its high impedance. If a line has a high voltage, it is 1 else it is 0. LineGraphCell also has a public function, called setColorbyValue(int). It gives the value field as an argument and set the color of the line object. If the value is undefined, i.e -1, the color is black. If it is 1,the color of line is green,and if it is 0 the color is red.

In addition to LineGraphCell class, ComponentGraphCell class has five fields and two functions. It has a type field which determines its type. For instance if the component is an AND gate the type is equal to the “and” string. The second field is a FileInputStream object, called configFile. As an input file, configFile is used for reading some necessary data. Every gate object has some numbers of input ports and output ports. Ports' type, i.e inputport or outputport, its value, i.e -1, 0 or 1, and their coordinate positions relative to the left top of the gate are the basic required data that will be read from the input file. By using these ports, system can connect a line to a component. For example, if a line is drawn, JGraph can detect that whether this line is connected to a port of a component by using coordinate axis positions of both line and component. If a line start point is connected to a port of a component, JGraph set the source of the line by this port. On the other hand, if a line end point is connected to a port of a component, JGraph set the target of the line by

this port. In addition to the configFile, there are ImageIcon and URL fields, calling jgraphIcon and jgraphUrl, respectively. These are two interrelated fields. The address of the icon is stored in the URL object and then it will be given to the constructor of the ImageIcon class as an argument. Namely, an image file in a specific directory will be created by using these fields. Also, there is another field, called isUserDefined. This field type is boolean, and it is used for determining whether the component object is a user defined or a default component.

As a result, draw class keeps a LineGraphCell vector and a ComponentGraphCell vector. Also it has another field, called graph. It is a JGraph object and it is used for keeping all drawn objects. In addition to its field, Draw class has four methods. DrawLine method is used for adding new line objects to the LineGraphCell vector and drawing all lines with two argument which types are Point2D and comes from the GUI class. The first Point2D object represents the start point of the line and the second one represents the end point of the line. In the same way, drawComponent method is used for keeping component objects into the ComponentGraphCell vector and drawing all component with two argument. But this time first one is a Point2D object and the second one is a String. The first argument keeps the coordinate axis position of the component and it is comes from the GUI class. The second argument keeps the type of the component. In addition to these draw methods, there are remove methods. RemoveLine method is used for removing any line from the drawing area by removing it from the LineGraphCell vector. In same way, the removeComponent method is used for removing a picked gate from the drawing area by deleting it from the ComponentGraphCell vector.

3.2.3 Class Diagram : Circuit Engine

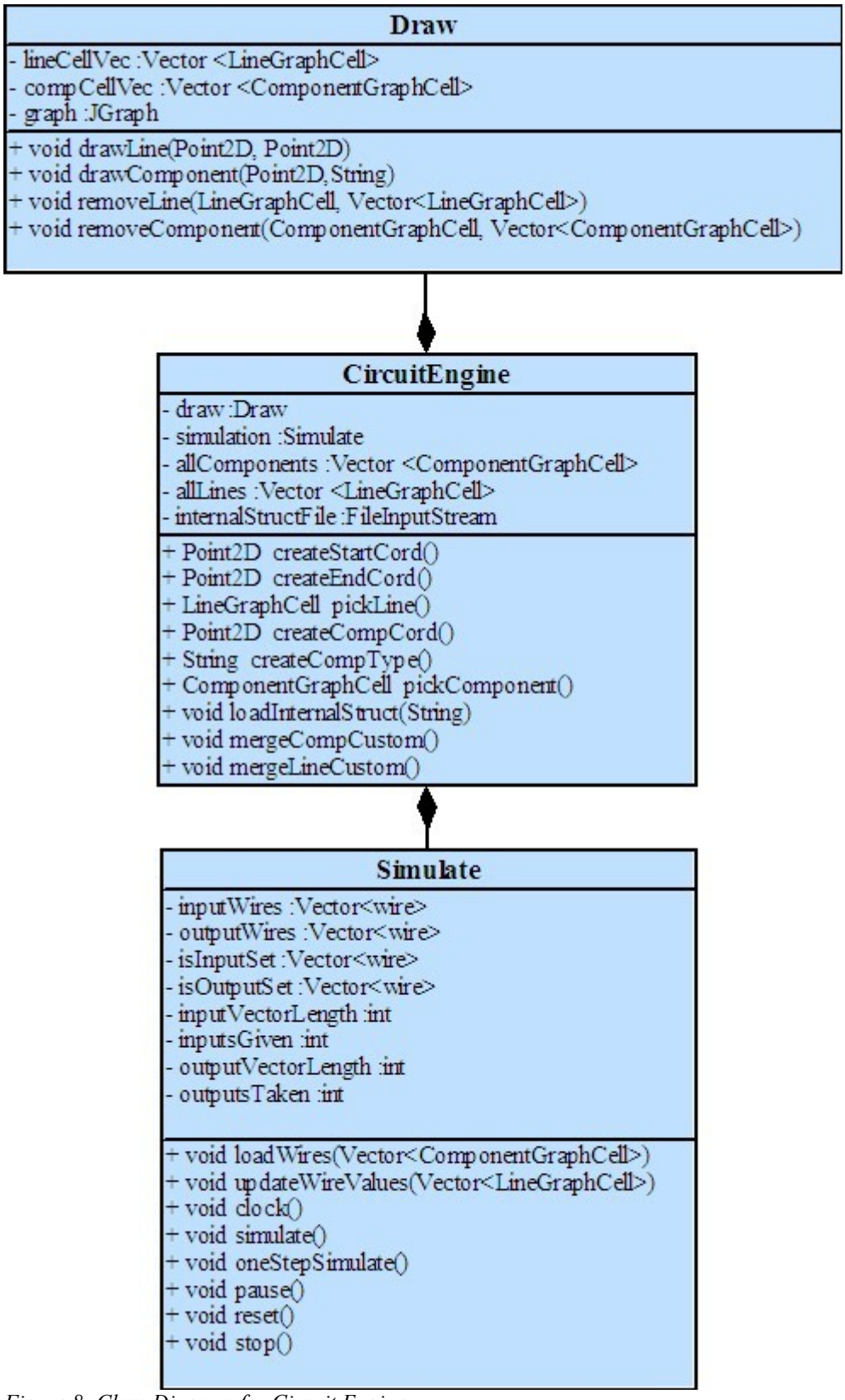


Figure 8: Class Diagram for Circuit Engine

CircuitEngine class has Draw and Simulate objects. The main task of the CircuitEngine class is the coordination of these two objects.

As explained before, Draw class will do all drawing process and will keep all information about the drawn objects. In order to simulate the design, some data transformation is necessary. Another task of the CircuitEngine class is to manipulate the data such that the design data will be used in simulation class.

The Draw class is responsible for all drawing operations. This class keeps two vectors, lineCellVector and compCellVector for lines and components respectively. The objects that are being drawn are kept in these vectors. These vectors keep LineGrapCell and ComponentGraphCell objects, all derived from DefaultGraphCell class of jgraph. LineGraphCell class keeps the drawn lines' data, color and value information. The ComponentGraphCell class keeps the component data in the same manner. This class keeps the components' image, configuration, origin and type data. Some of these datum are required for converting the drawing into some structure that JHDL understands.
At each mouse event that is related to drawing will invoke this class' methods. At each object creation, the object will be given to the jgraph instance and copied to the related vector for transferring the circuit data to the simulation engine.

CircuitEngine class is the connector class between the Draw class and Simulate class. The draw class will supply this class the line and component vectors. Then CircuitEngine class will look for user created components in the given component vector and replace them by the unserialized vectors of the user created components. MergeCompCustom, MergeLineCustom will merge the internal structure of this custom components with the vectors in the draw object. After all new vectors will be created which will be necessary for the JHDL to simulate the design. This replacement will not affect the visualization of the components. It is done for only simulation.

Simulate class will do the main simulation process. The TestBench interface is a top level cell for generating test data to drive a circuit. Logic class provides many convenient methods for accelerating structural design. For example, the method call `and(a, b)` instantiates a new 2-input and gate automatically, wires up a and b to the inputs, instantiates a new wire and then connects the new wire to the output of the gate, and returns the new wire. Because these method calls return wires, not gates, it allows to have nested method calls to quickly build up complex logic circuits.

The CircuitEngine will pass the line and component vectors to the Simulation class as arguments to load methods. The class, then, will keep these vectors in internal variables. The simulation methods (simulate, oneStepSimulate, etc.) will then use these variables and do the simulation by calling JHDL simulation methods.

3.2.4 Class Diagram : Scripting

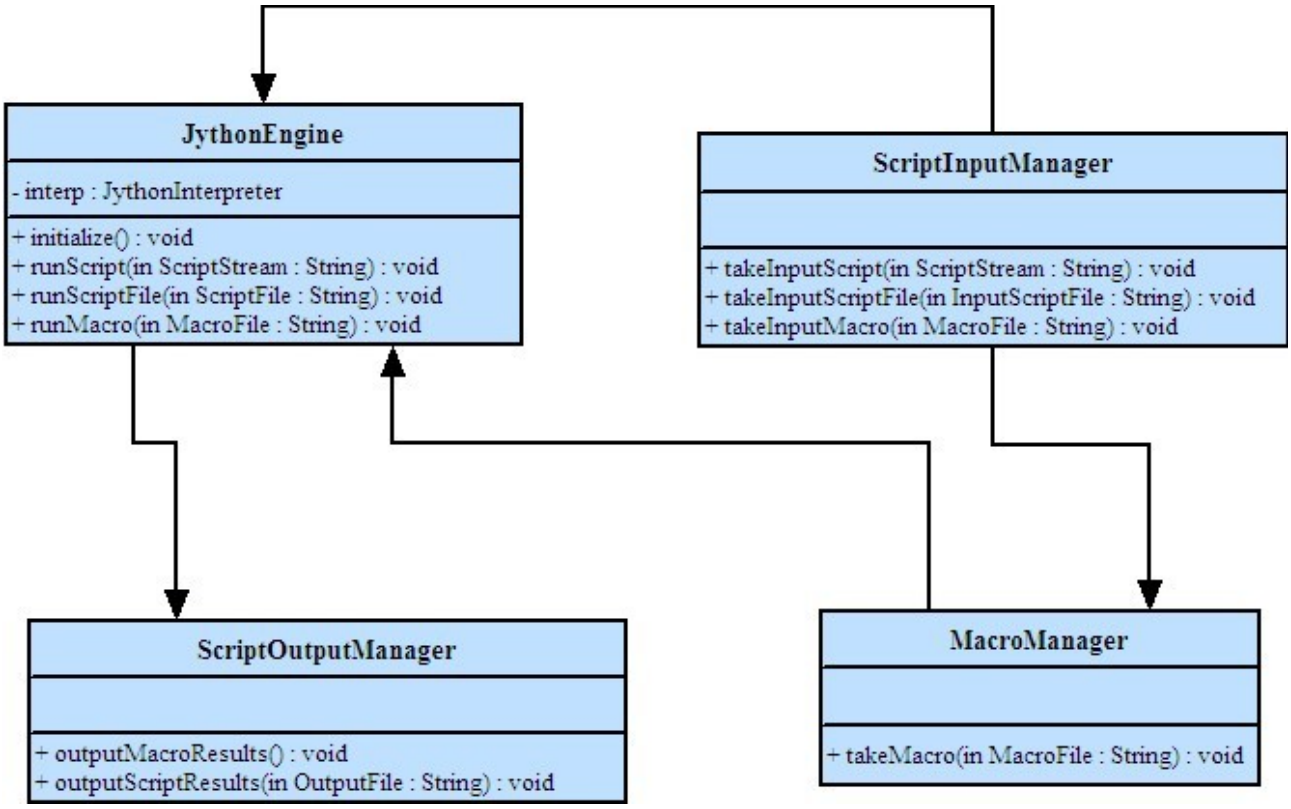


Figure 9: Class Diagram of Scripting Module

The scripting module runs in the following order: The user inputs some script (either by the scripting console in the GUI or by opening a script file) and the GUI directs this script to the ScriptInputManager class. The ScriptInputManager has 3 methods to handle the incoming script. After getting the script input, this class calls respective methods from the JythonEngine class. This class does the main Python execution. After evaluating the script commands, JythonEngine class calls the ScriptOutputManager to display the outputs. Then, the ScriptOutputManager calls relevant GUI methods to display the results.

JythonEngine has an external module connection to the Simulation Module. If the script contains some simulation control commands, the JythonEngine directly calls the Simulation Control Methods. The results of this execution is then handled by the Simulation Module Classes.

Execution of macros are treated as simplified script commands. Macros have limited power relative to scripts. The user initiates definition of the macro by GUI events and the MacroManager handles these events and converts to Python script. Then, the macro is processed as a script file.

3.2.5 Class Diagram : GUI and File Operations

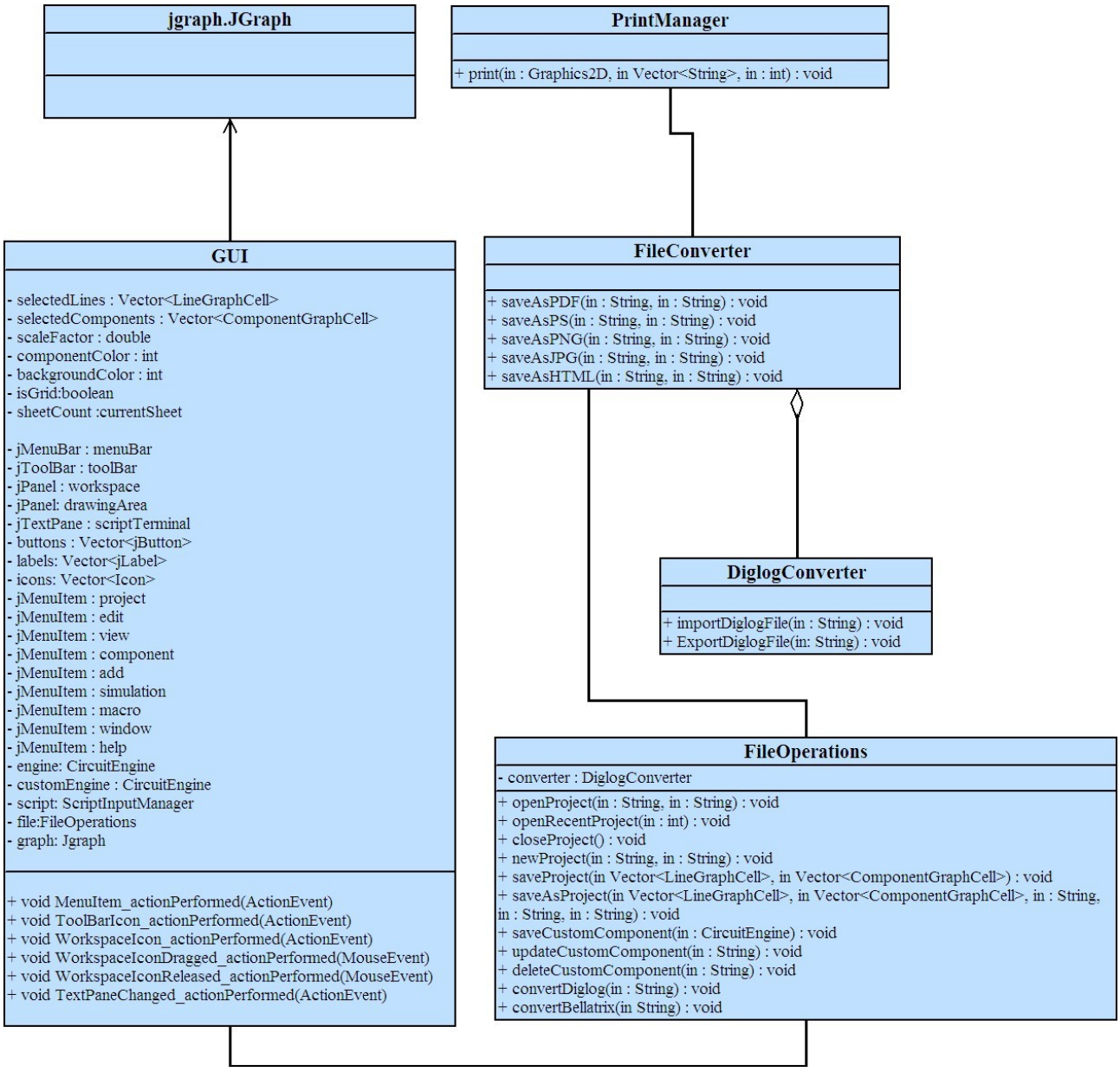


Figure 10: Class Diagram for GUI and File Operations

In order to generate the GUI stated at [SECTION 4], some components of javax.swing package will be used in the GUI class such that a MenuBar, a ToolBar, two Panels, a TextPane and a number of buttons, labels and icons. First Panel represents the WorkSpace Pane and the second one represents the Drawing Area. The buttons, labels and icons will be used in the Tool Bar, Menu Bar and the Workspace Pane in order to relate each component to a button, label or icon where necessary.

GUI class also encapsulates two CircuitEngines, a ScriptInputManager, a FileOperations and a JGraph object. First circuitEngine object will be used for the simulation of the whole circuit.

Second circuitEngine object will be used for the simulation of the custom defined circuit. The ScriptInputManager object will be used for performing script operations. The FileOperations object will be used for all file operations such as saving the project, printing a document, converting a file. The Jgraph object will be used for operations such as copy, paste, zoom in which will be explained more detailed in the following paragraphs.

Finally there are certain fields in GUI class, namely selectedLines ,selectedComponents, scaleFactor, componentColor, backgroundColor, isGridcurrentSheet. These fields are required to pass arguments to JGraph object in order to perform GUI actions such as cut, copy, paste, undo, redo which again will be explained in the following paragraphs.

GUI class basically provides a way to perform operations that require user interaction. For example selecting a Menu Item, clicking on a ToolBar icon, dragging a Component from the Workspace Pane to the Drawing Area, moving a component in the Drawing Area, entering text in the Script Terminal.

Each action stated above triggers an event indicating that the user demands an operation. These events are stated as follows in the class diagram:
MenuItem_actionPerformed(ActionEvent)
ToolBarIcon_actionPerformed(ActionEvent)
WorkspaceIcon_actionPerformed(ActionEvent)
WorkspaceIconDragged_actionPerformed(MouseEvent)
WorkspaceIconReleased_actionPerformed(MouseEvent)
TextPaneChanged_actionPerformed(ActionEvent)

Notation:

MenuItem_actionPerformed(ActionEvent)
ToolBarIcon_actionPerformed(ActionEvent)

- "MenuItem" represents all possible Menu Items of Bellatrix such as jMenuProjectSave, jMenuSimulationRun. Similarly "ToolBarIcon" represents all possible ToolBar icons of Bellatrix such as a run icon, a pdf icon and so on.
- ActionEvent can represent either choosing a Menu Item or selecting a ToolBar icon.

WorkspaceIcon_actionPerformed(ActionEvent)
WorkspaceIconDragged_actionPerformed(MouseEvent)
WorkspaceIconReleased_actionPerformed(MouseEvent)

- "WorkspaceAreaIcon" represents all possible DrawingArea Icons of Bellatrix such as an And Gate icon or a Clock icon.

- ActionEvent represents choosing a Workspace icon
- MouseEvent can either represent dragging the mouse or releasing it.

TextPaneChanged_actionPerformed(ActionEvent)

- "TextPane" represents the script terminal.
- ActionEvent represents entering a text to the script terminal.

Each icon representing a component or a wire in the Drawing Area is called a DrawingAreaIcon such as an icon representing an And Gate. The drawingArea icons are defined as the same type with the WorkspaceArea icons. They are simply an ImageIcon object defined in javax.swing package**.** Workspace icons are picked when the user triggers the WorkspaceIcon_actionPerformed(ActionEvent) event by clicking on the icon from the Workspace Pane. Workspace icons can be dragged to the Drawing Area by the user and after that they are treated as DrawingArea icons. This is accomplished by the WorkspaceIconDragged_actionPerformed(MouseEvent) and WorkspaceIconReleased_actionPerformed(MouseEvent) methods.

Also the user can select and drag the components (which are actually ImageIcons representing gates, clock etc) in the drawing area which are accomplished by the the native methods of the Jgraph object.

The sequence proceeds as follows: Each time the user triggers an event from the MenuBar (by choosing a MenuItem), Toolbar (by clicking on a ToolBar Icon), TextPane (by entering text in Text Pane) corresponding method of the related object (Circuit Engine, ScriptInputManager, FileOperations, GUIActions) will be called and that method will handle the operation.

For example when the user selects the Save Option of the Project Menu, the jProjectSave_actionPerformed(ActionEvent) event will be triggered and inside the event, the saveProject method of the fileOperations object will be called in order to perform the save operation.

Similarly when the user selects the Copy Option of the Edit Menu, the jEditCopy_actionPerformed(ActionEvent) event will be triggered and inside the event, the copy method of the jGraph object will be called in order to perform the copy operation.

A full list of used JGraph methods are stated as follows:

undo()
redo()
group(Object [])
ungroup(Object [])
showGrid()
removeGrid()
changeBackgroundColor()
selectAll()
deselect()
copy(Action)
cut(Action)
paste(Action)
setScale(double)

The methods are included in the JGraph library and used by the GUI class. Detailed examples including the interaction between these classes will be explained in the sequence diagrams [SECTION 3.3]
The file operations are intuitive. The FileOperations class handles the opening and saving file operations. Saving the file in the .bx format will be done directly in this class.
Converting options will be handled by the subclasses of this class, FileConverter and DiglogConverter. FileConverter class will be capable of converting the drawing to several formats, including PS and PDF. This is done by corresponding methods in the FileConverter class. When the user selects the Save As option from the Project Menu, a Save Dialog will be opened and the user

will choose to save the file in either ps, pdf, png, jpeg, html formats. According to the choosen format, the corresponding method of the FileConvertor class will be called.
For example if the user chooses to save as pdf format from the Save Dialog, the saveAsPdf method of the FileConverter class will be called.

Converting to Diglog format is handled by DiglogConverter class. This class takes the circuit data and converts it to Diglog format. This class also responsible for importing Diglog files. The Diglog format file is then converted to our format in the same way stated for saving as pdf format.

3.3 Sequential Diagrams

3.3.1 Circuit Design Module

3.3.1.1 Line Operations & Component Operations

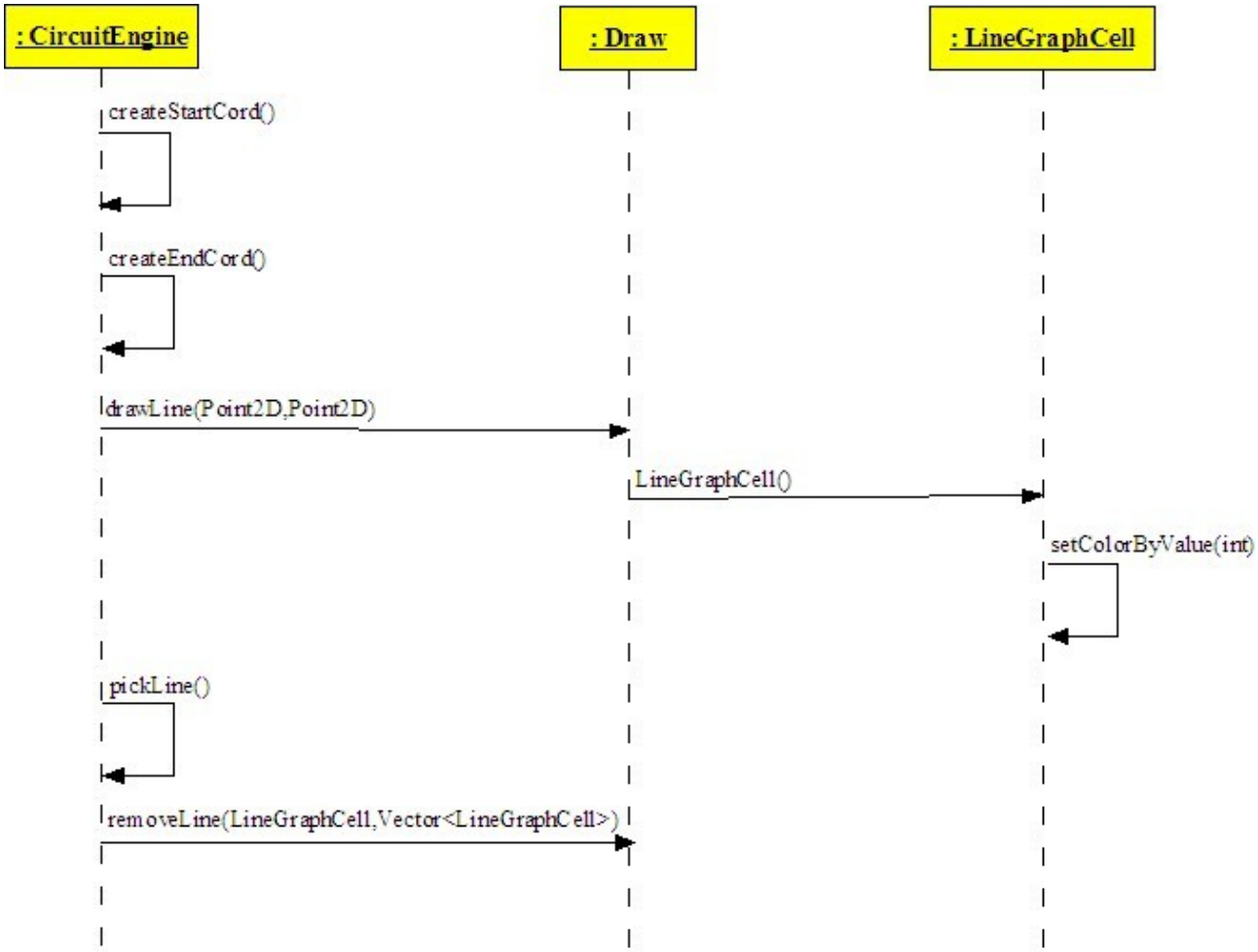


Figure 11: Sequential Diagram of Line Operations

Line operations will be done by three classes, CircuitEngine, Draw and LineGraphCell. In same way, component operations will be done by CircuitEngine, Draw and ComponentGraphCell. CircuitEngine holds a Draw object and Draw holds a LineGraphCell or ComponentGraphCell object. The mouse events return the start and end points of the line (start point: the point where the mouse is pressed; end point: the point where the mouse is released), or the origin of the component. If the object is a component, it also holds the type of the component. Then, CircuitEngine calls the draw methods of the Draw class. Draw class creates a LineGraphCell or ComponentGraphCell object and draws a line/gate on the circuit drawing area. If the object is LineGraphCell, it can set its color according to its value, else if the object is ComponentGraphCell, it reads the configuration file

from the disk and loads the corresponding image. In addition to drawing, CircuitEngine can control the deletion of the objects. Thanks to Jgraph, if the mouse clicks on a line or a gate, the object will be selected. Selected item will be deleted from the LineGraphCell or ComponentGraphCell vector by calling the remove method of the Draw class. Therefore, the selected object will be removed from the circuit drawing area.

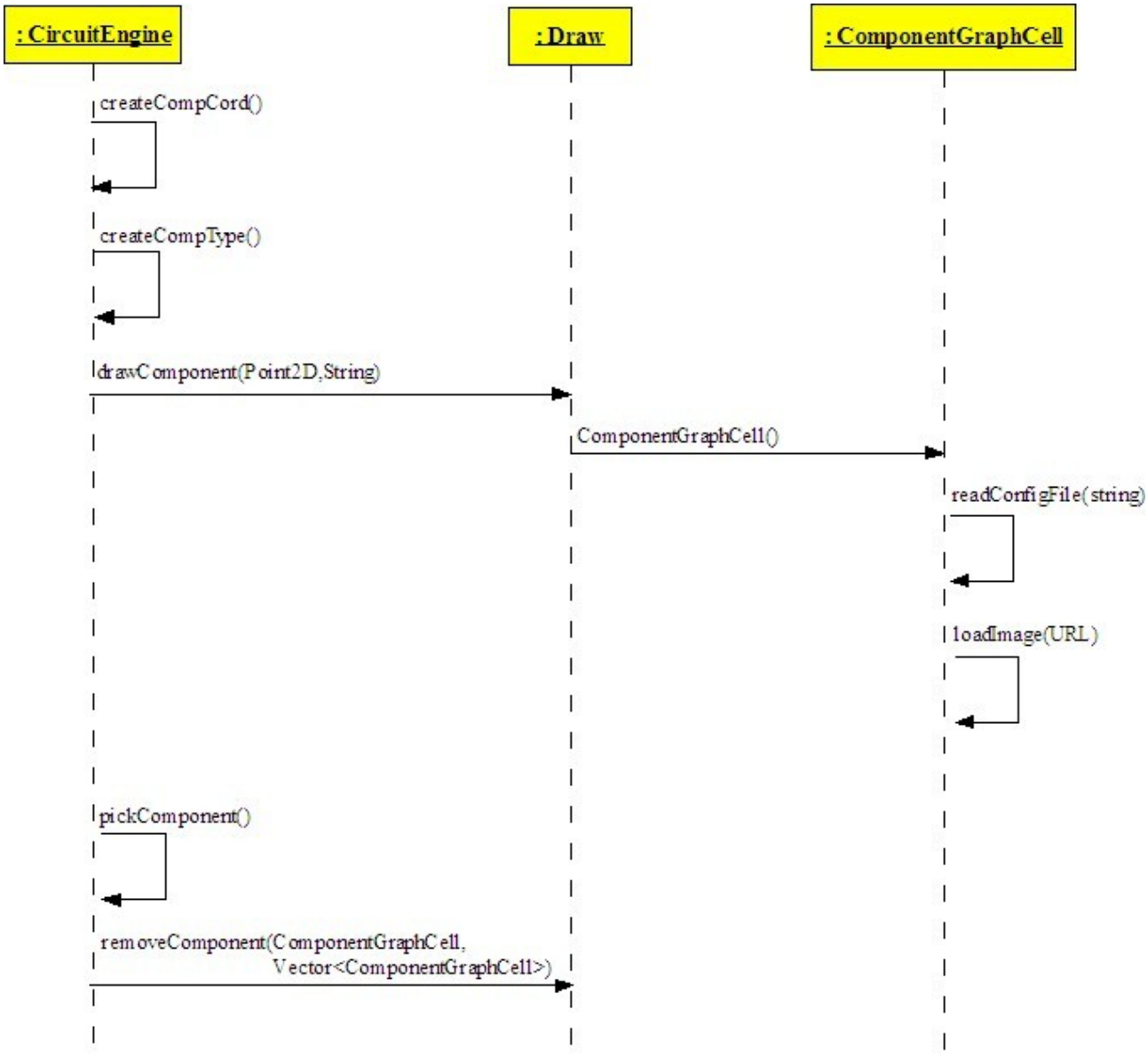


Figure 12: Sequential Diagram of Component Operations

3.3.1.2 Create Custom Component

The creation of a custom component will proceed as follows:

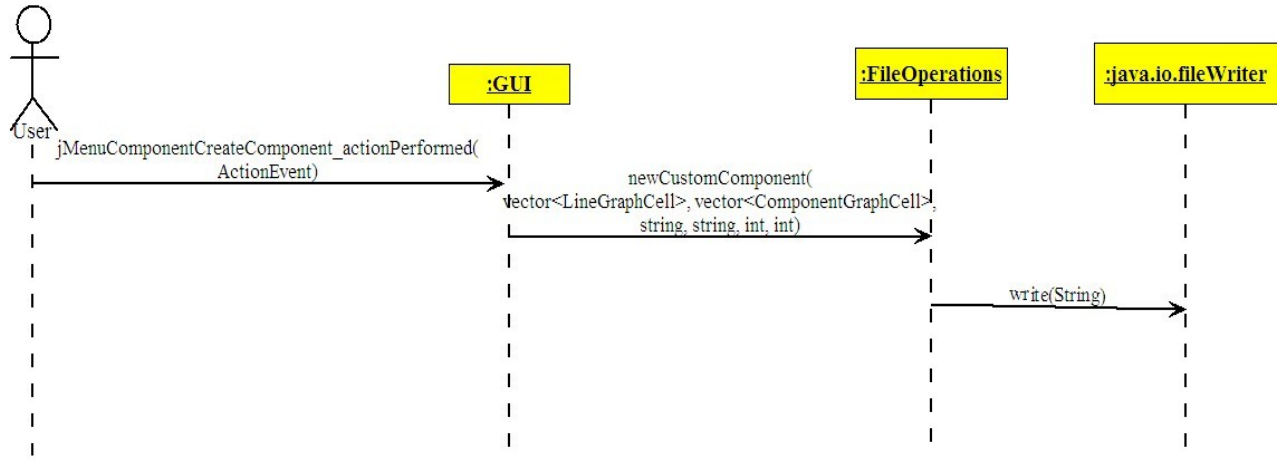


Figure 13: Sequence Diagram for Create Custom Component

When the user selects the Create Component option from the Component Menu, an jMenuComponentCreateComponent_actionPerformed(ActionEvent) event will be triggered informing the GUI that a user demanded to create a custom component. After that, GUI class will call the newCustomComponent method of its FileOperations object. Finally FileOperation object will simply use the write method of the fileWriter class of java.io package in order to save the Custom component in a file.

3.3.2 Simulation Module

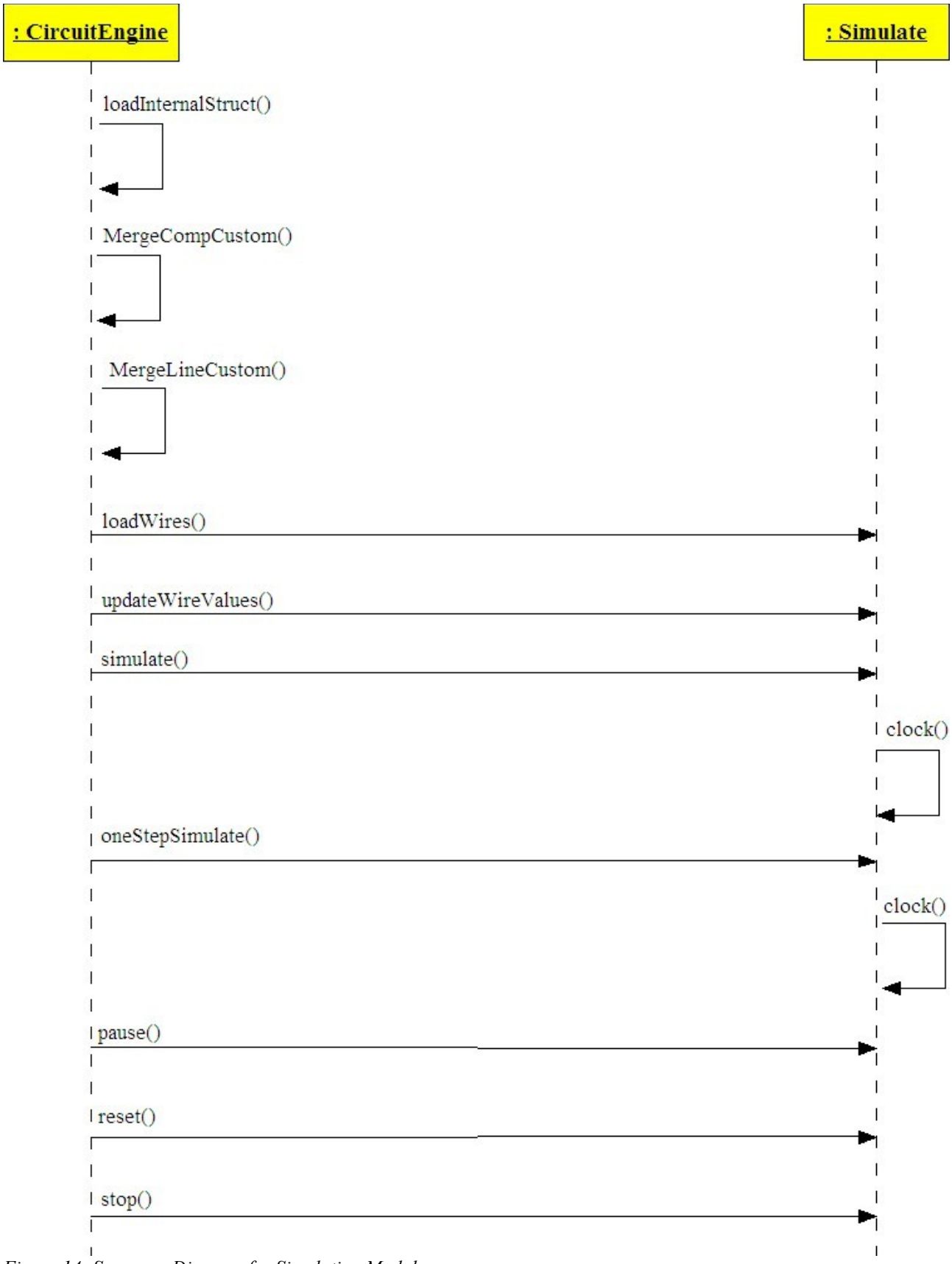


Figure 14: Sequence Diagram for Simulation Module

Simulation will be done by two classes, CircuitEngine and Simulate. Before staring the simulation, CircuitEngine must prepare the two vectors: line vector and component vector which Simulate class will use for simulation. CircuitEngine does this job by loading the internal structure of the user defined components used in the design and then merges their functionality (lines and components defined in them) with the other components and lines. After all this pre-process Simulate object starts simulation by calling the simulation functions of the JHDL. Simulate has a clock() method which overrides the clock() method of JHDL. All value operations (giving new values to wires) done in clock() method. JHDL calculate the new values of the wires step by step. Therefore simulate method of Simulate object calls the clock() method, until simulation finishes; however the oneStepSimulate method calls the clock() method only one time. Also JHDL considers the propagation delay of components. Other options of the simulation like pause, stop, reset is done again by using the methods of JHDL defined in its TestBench interface.

3.3.3 File Operations Module

3.3.3.1 Save/Load

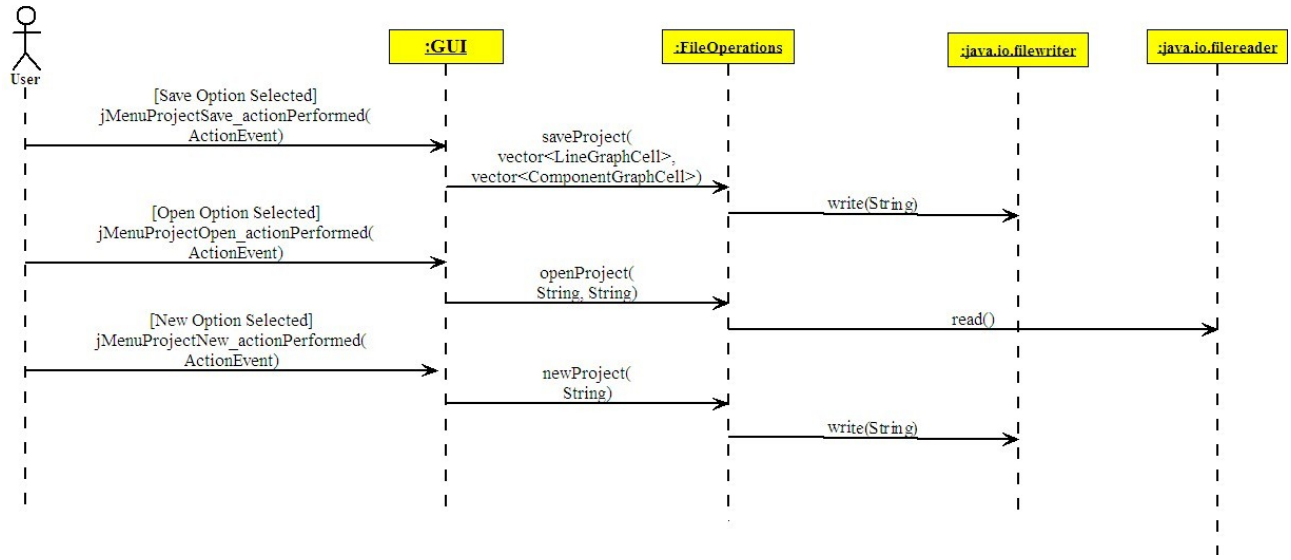


Figure 15: Sequence Diagram for Save/Load

Save/Load Operations will proceed as follows:

When the user selects the Save Option from the Project Menu, an jMenuProjectSave_actionPerformed(ActionEvent) event will be triggered informing the GUI that the user demanded to save the project. After that, GUI class will call the saveProject method of its FileOperations object. Finally the FileOperations object will simply use the write method of the fileWriter class of java.io package in order to save the project in a file.

When the user selects the Load Option from the Project Menu, an jMenuProjectOpen_actionPerformed(ActionEvent) event will be triggered informing the GUI that the user demanded to open a project. After that, GUI class will call the openProject method of its FileOperations object. Finally the FileOperations object will simply use the read method of the fileReader class of java.io package in order to open the project.

When the user selects the Open Option from the Project Menu, an jMenuProjectNew_actionPerformed(ActionEvent) event will be triggered informing the GUI that the user demanded to create a new project. After that, GUI class will call the newProject method of its FileOperations object. Finally the FileOperations object will simply use the write method of the fileWriter class of java.io package in order to create a new project.

3.3.3.2 Print

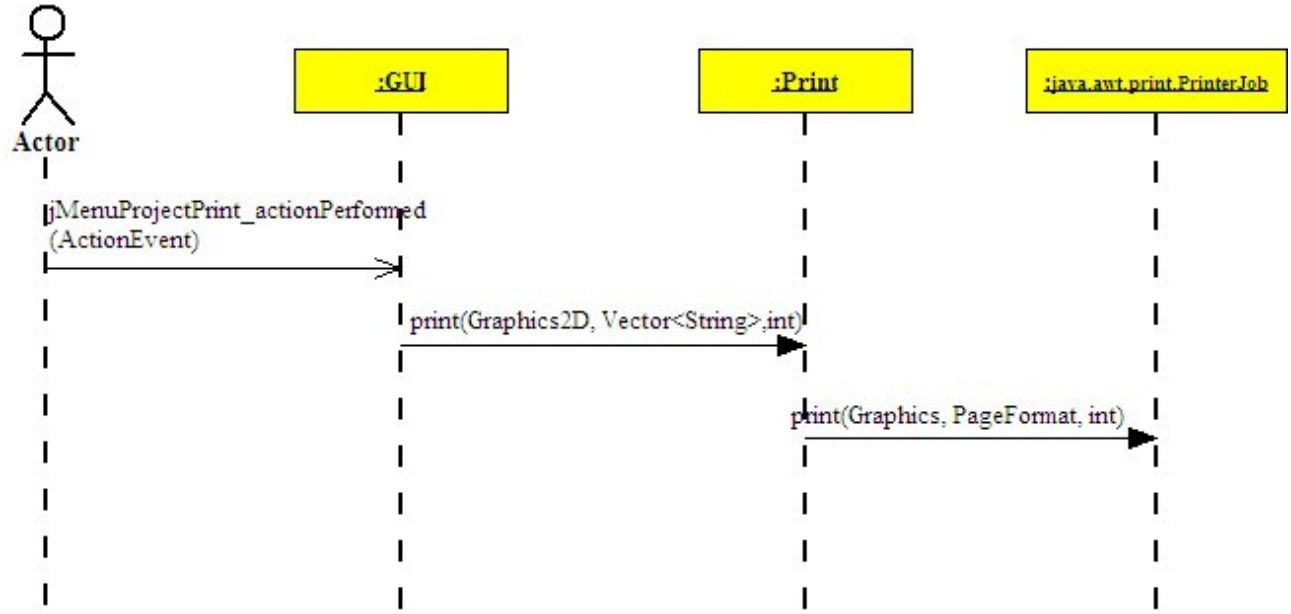


Figure 16: Sequence Diagram for Print

As it can be seen from the diagram, GUI class calls the related function of Print class when the user triggers an ActionEvent by selecting the corresponding option from the menu bar. Print class uses the java.awt.print.PrinterJob library of Java to locate a service which can export 2D graphics to a stream as Postscript. This may be spooled to a Postscript printer, or used in a postscript viewer.

3.3.3.3 File Converter

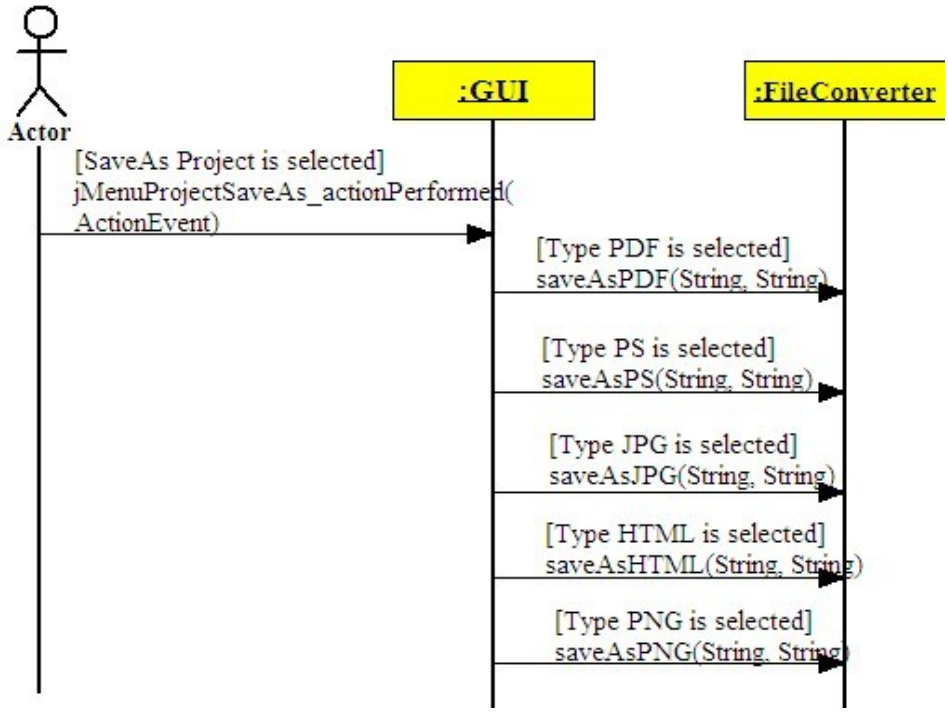


Figure 17: Sequence Diagram for File Converter

A File Conversion Operation will proceed as follows:

When the user selects the Save As Option from the Project Menu, an jMenuProjectSaveAs_actionPerformed(ActionEvent) event will be triggered informing the GUI that the user demanded to save the project in a printable file. After that, FileOperations object of the GUI class will call the one of the SaveAsPDF, SaveAsPS, SaveAsJPG, SaveAsPNG, SaveAsHTML methods of FileConverter class . Finally the FileConverter class will perform the neccessary operations to save the project in the specified format.

3.3.4 Script Module

3.3.4.1 Script Operations

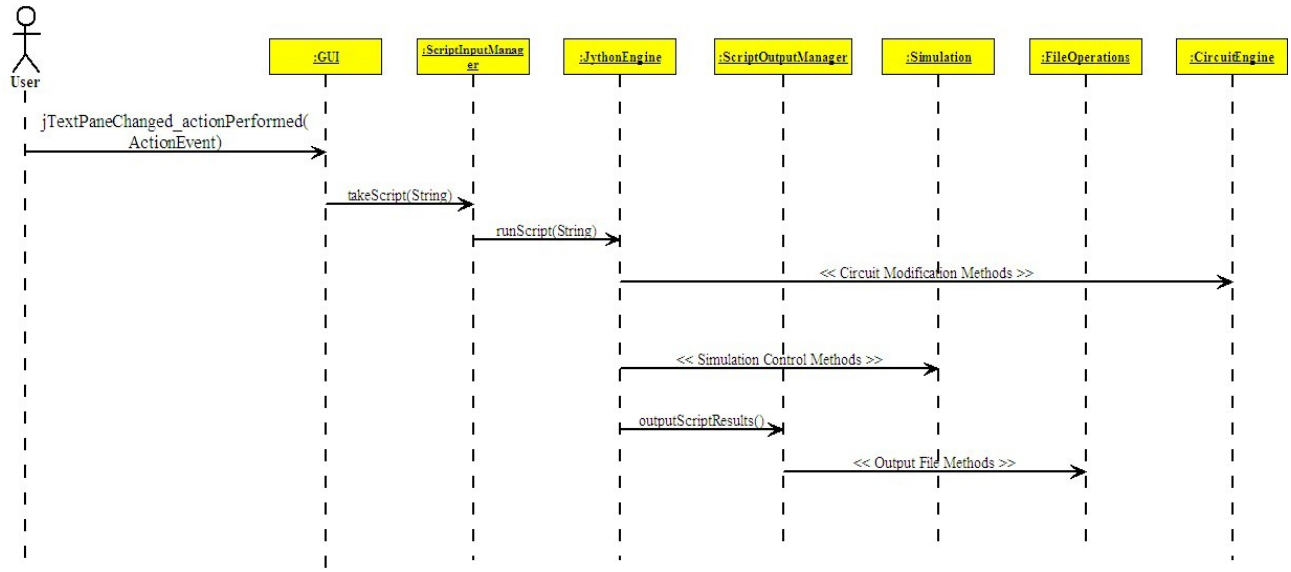


Figure 18: Sequence Diagram for Script Operations

When the user inputs a script from the script console, the system forwards this input to ScriptInputManager. This class pre-processes the script and then invokes the JythonEngine class' runScript method. This method executes the script by instantiating a JythonInterpreter object. This object now controlls the script execution. This way, the script may control the Simulation class and/or CircuitEngine class.

3.3.4.2 Macro Operations

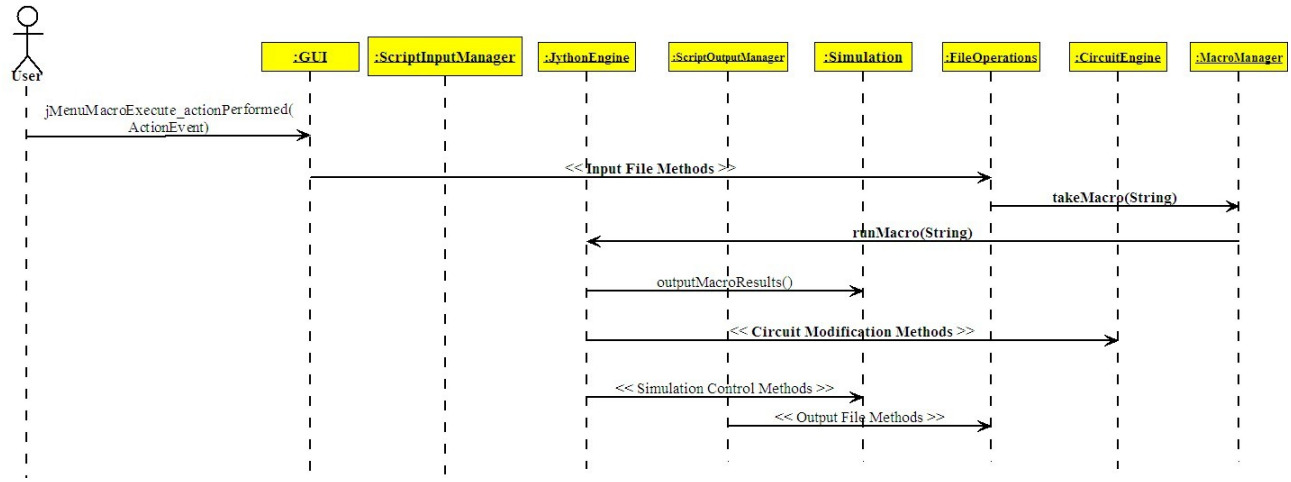


Figure 19: Sequence Diagram for Macro Operations

Running a macro is very similar to running a script in the internal structure. This is because macros are actually scripts. The macro is loaded from the file and pre-processed by the MacroManager class. This class then passes the macro to the JythonEngine and the rest is the same as script execution process.

4 GUI Design

4.1 Bellatrix Overview

Bellatrix EDA is based on a MDI concept (Multiple Document Interface). Several sheets can be used to draw the schematics and simulate them.

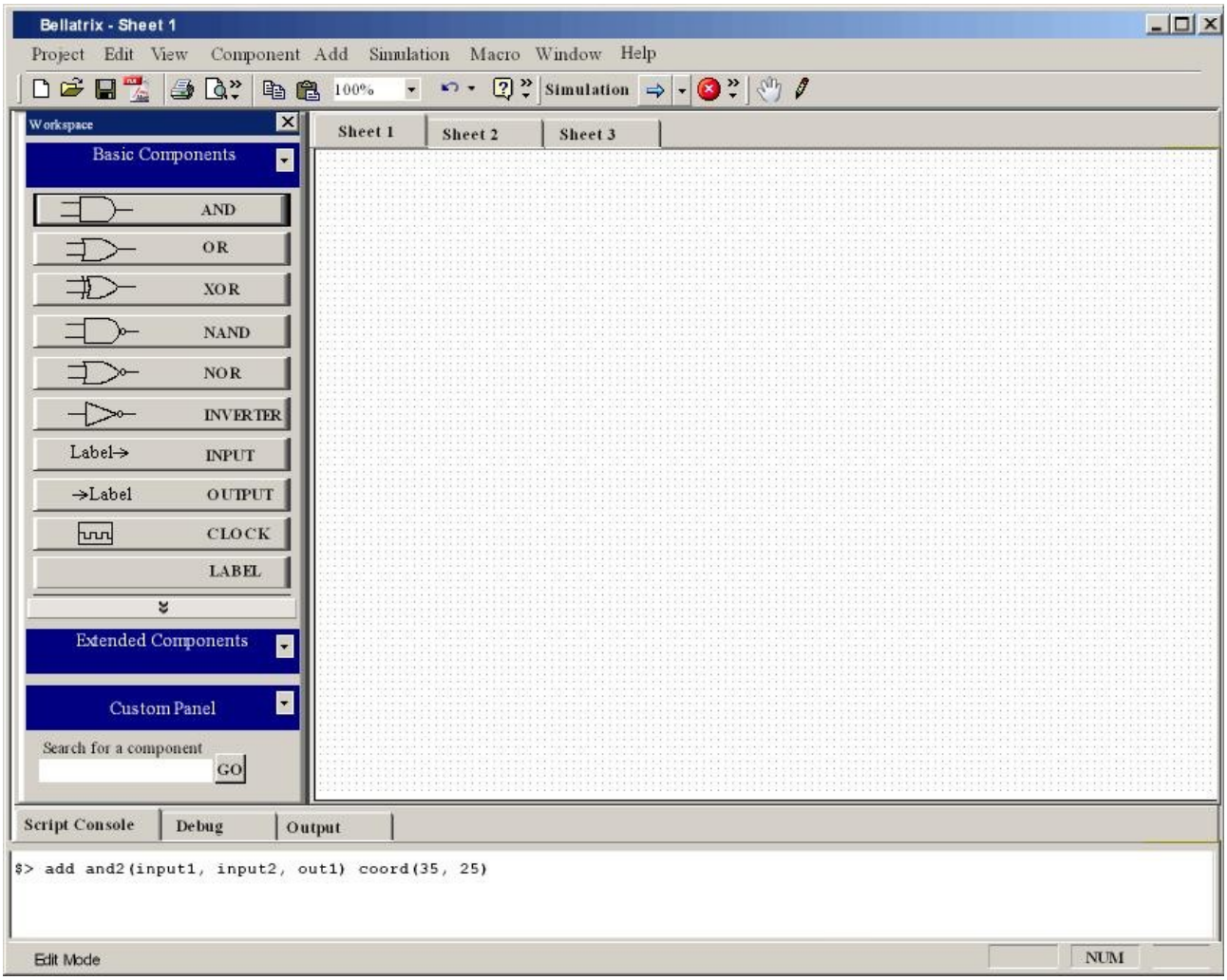


Figure 20: The theoretical view of the GUI. Changes may be applied due to differences in Java window interface.

Project title header contains the name of the application and the current active sheet.

Figure 21: Title Bar of Bellatrix.

Menu bar allows to access all system features of the application.

Figure 22: Menu Bar of Bellatrix.

Tool bar contains the symbols of most frequently used features.

Figure 23: Tool Bar of Bellatrix.

Workspace view displays the components that are available.

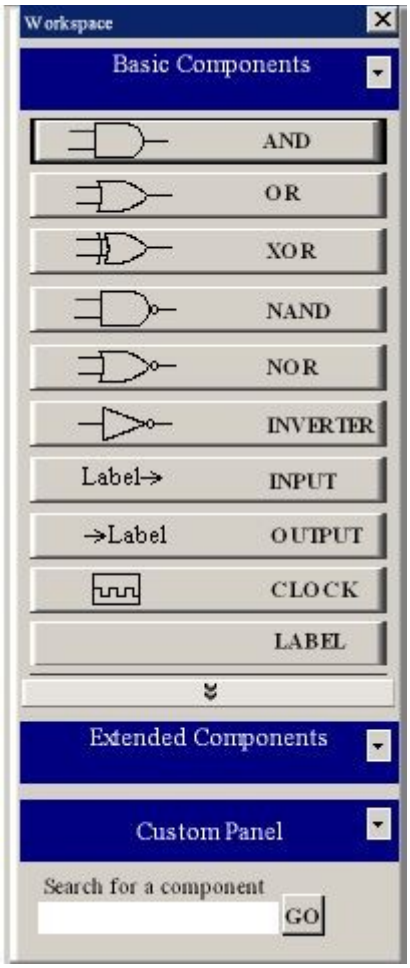


Figure 24: Workspace: Standard and User-Defined Components.

Console view is used to display informations related to the Edit and Simulation modes.

Figure 25: Console: The general input-output area of the system.

Status Bar displays some interesting informations like the current cursor position in file when the user edits a script file or the system clock.

Figure 26: Status Bar shows the current status of the system.

Drawing Area is the place where the circuit is drawn. This place can support various pages called "sheets".

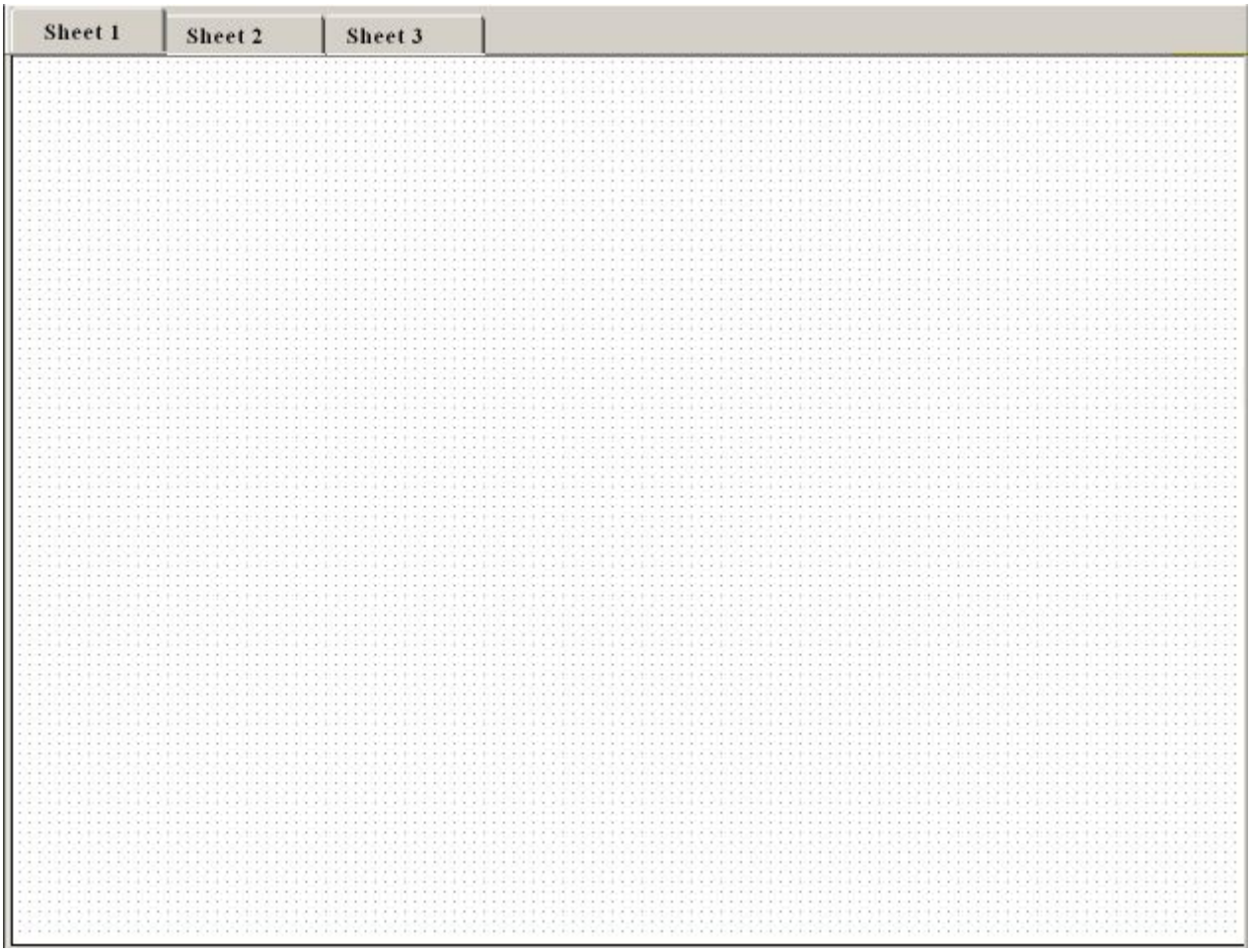


Figure 27: Main Drawing Area

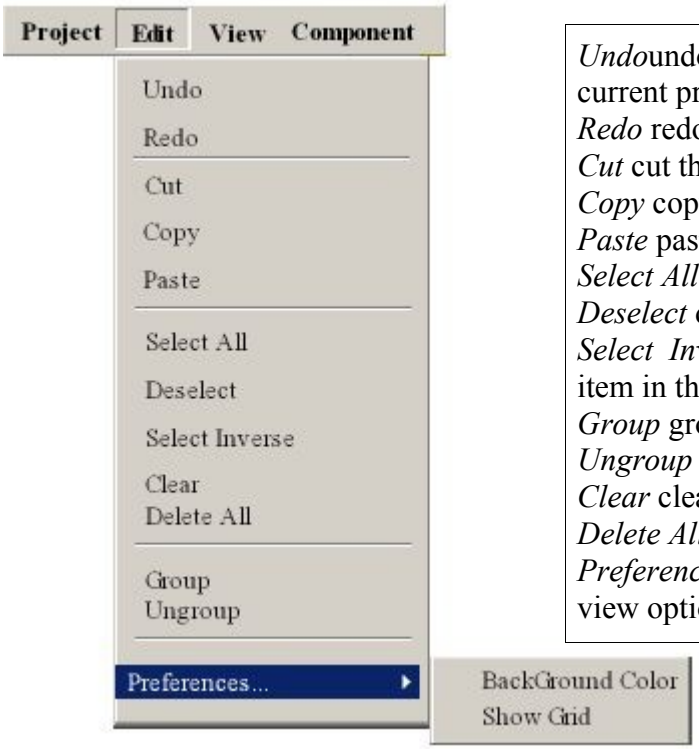
4.2 Menus

4.2.1 Project Menu

New reate a new project.

Open open an existing project.
Close close the current project.
Save save the current project.
Save as save the current project with a different name.
Print prints the current project.
Print Preview enables the user to see the print format of the current project.
Print Setup enables the user to view and change the print settings
Recent Files enables the user to view and open the most recently used files
Exit enables the user to exit from the program.

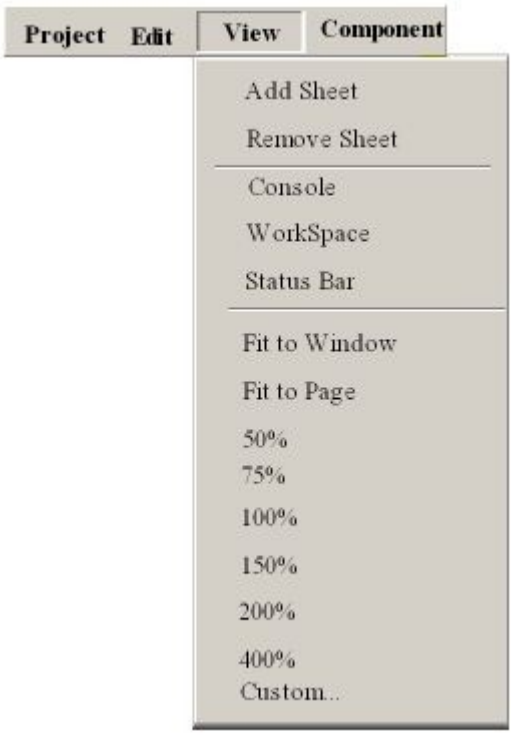
4.2.2 Edit Menu



*Undo*undo the last action before the user save the current project.
Redo redo until the first undo action.
Cut cut the selected item(s).
Copy copy the selected item(s).
Paste paste the last cut or copied item(s).
Select All select all items in the current sheet.
Deselect deselect a selected item.
Select Inverse select all items except the selected item in the current sheet.
Group group the selected items.
Ungroup ungroup the selected group.
Clear clear the selected items.
Delete All delete all items in the current sheet.
Preferences contains the background color and grid view options.

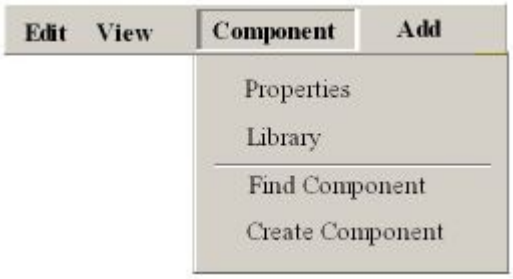
Background color provides a color palet to the user in order to set the background color of the drawing area. Default color is white.
Show Grid provides a grid view.

4.2.3 View Menu



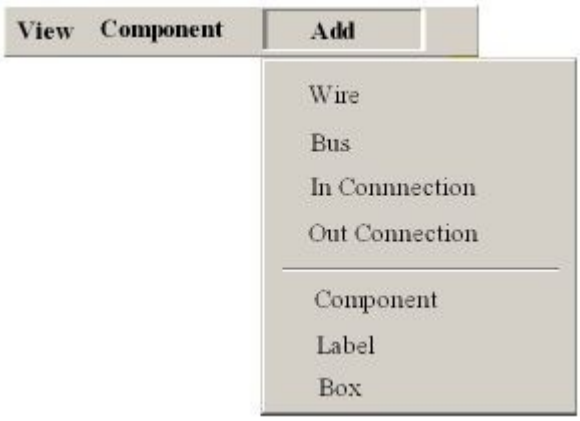
Add Sheet insert a new sheet to the current project.
Remove Sheet option shall enable the user to remove the current sheet from the project.
Console option shall enable the user to show or hide the console view.
Workspace option shall enable the user to show or hide the workspace view.
Status Bar option shall enable the user to show or hide the status bar.
Zoom options shall be listed such that *Fit to Window, Fit to Page, 50%, 75% ..., Custom.*

4.2.4 Component Menu



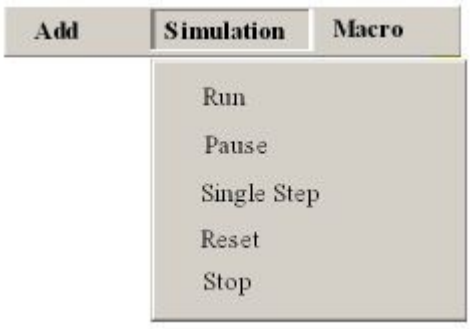
Properties display the name, code and input/output informations of the selected component.
Library display the library pages which contains all the components that are available as listed.
Find Component display a pop up window which contains a text box for the name of the searched component and enable the user to find it.
Create Component adds a new component defined by the user.

4.2.5 Add Menu



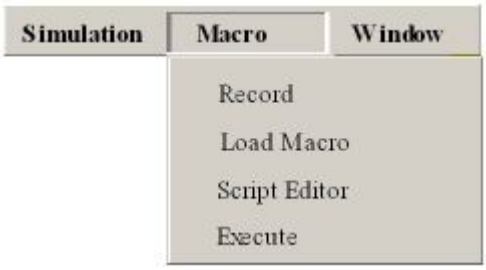
Wire connect components with wires.
Bus option add bus connections.
In Connection add input connection instead of line drawing between sheets.
Out Connection add output connection instead of line drawing between sheets.
Component display the basic components panel in the workspace.
Label add label.
Box draw boxes around the circuits.

4.2.6 Simulation Menu



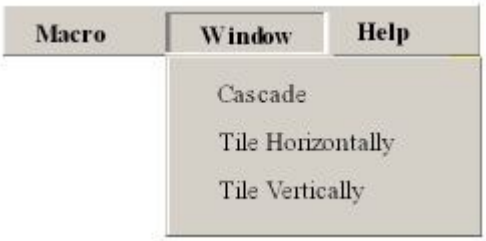
Run start the simulation of the current project. All the following commands in this menu shall be selectable after the simulation is started by run command.
Pause suspend the simulation of the current project.
Single Step perform the simulation step by step and showing the internal steps
Reset restart the simulator.
Stop exit from the simulation of the current project and return to the edit mode.

4.2.7 Macro Menu



Record save the current project as a macro in order to support reusage of the drawing.
Load Macro load a previously recorded macro.
Script Editor open a text file with the file extension .bx in order to enable the user to write or edit a script file. These scripts can be used for testing.
Execute execute the selected script file.

4.2.8 Window Menu



Cascade cascade the sheets of the current project.
Tile Horizontally tile the sheets of the current project horizontally.
Tile Vertically tile the sheets of the current project vertically.

4.2.9 Help Menu

About connect to the our web site in order to give information about the Project Bellatrix.
Language Help display the tutorials about the scripting language.
Custom Help display the tutorials about the usage of the tool.

5 Features

5.1 Custom Component Creation

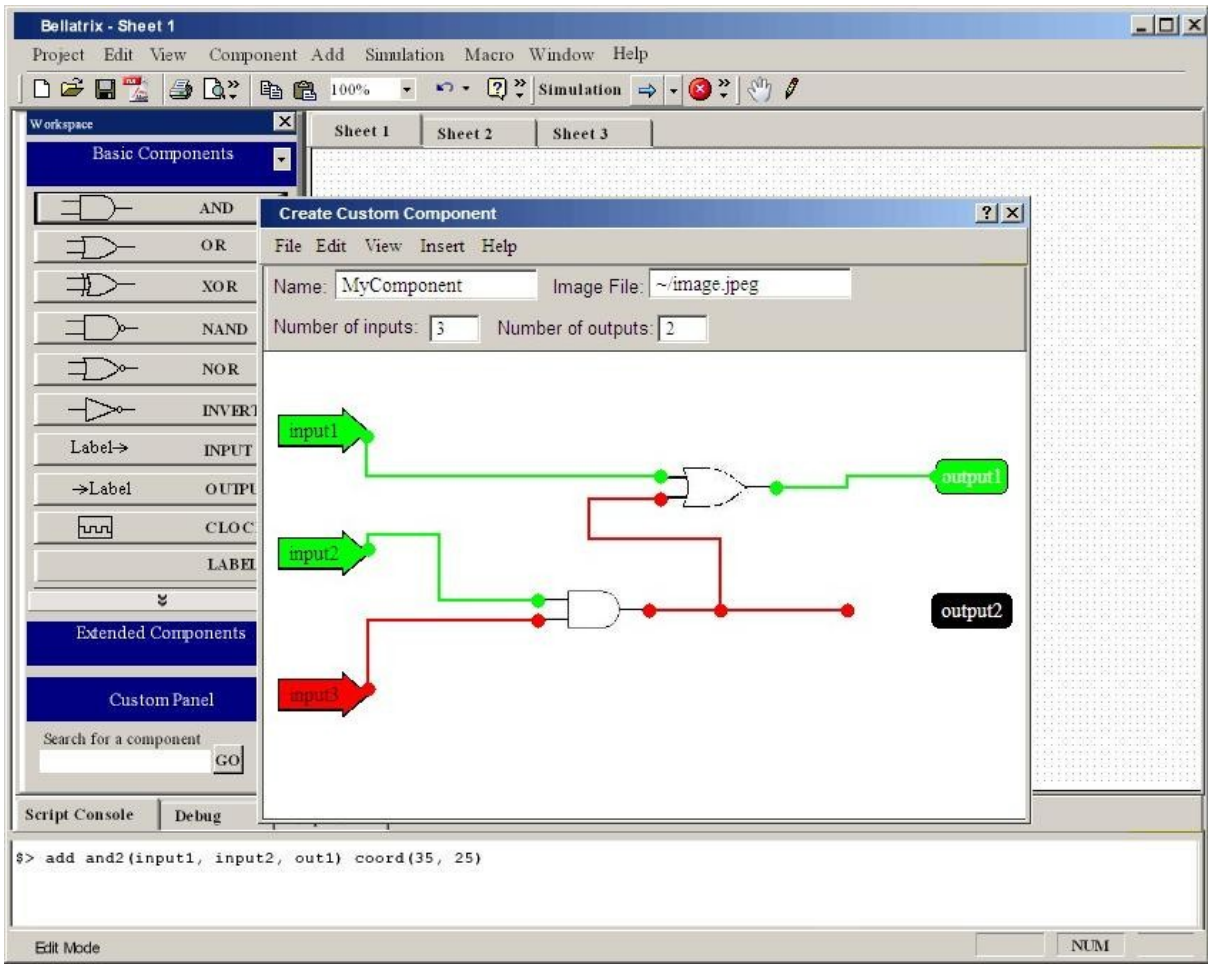


Figure 28: Pop-up window for creating a new component

As we wrote in our previous reports, the user will be able to create new components. This will be done by "Create Custom Component" pop-up window. The idea is that the user will define the behaviour of the circuit by just drawing it as a circuit.

In this window, the user will supply the name, image file and number of input and output ports into corresponding text boxes. As soon as the user gives the number of ports, the input and outputs of the component will be created and displayed on the drawing area. After this, the user will draw the component's behaviour as a circuit. Then, the user will save the component using the file menu. The component will be saved under components/custom directory in serialized class form. Configuration and image of the component will be saved under the same directory, all having the name given by the user in the window.

The component will be verified before being saved. If there are any unconnected components, and such other errors, the application will give an error.

From the file menu, the user will be able to open and edit an existing custom component.

A detailed help and tutorial will be supplied with the application.

5.2 Directory Structure

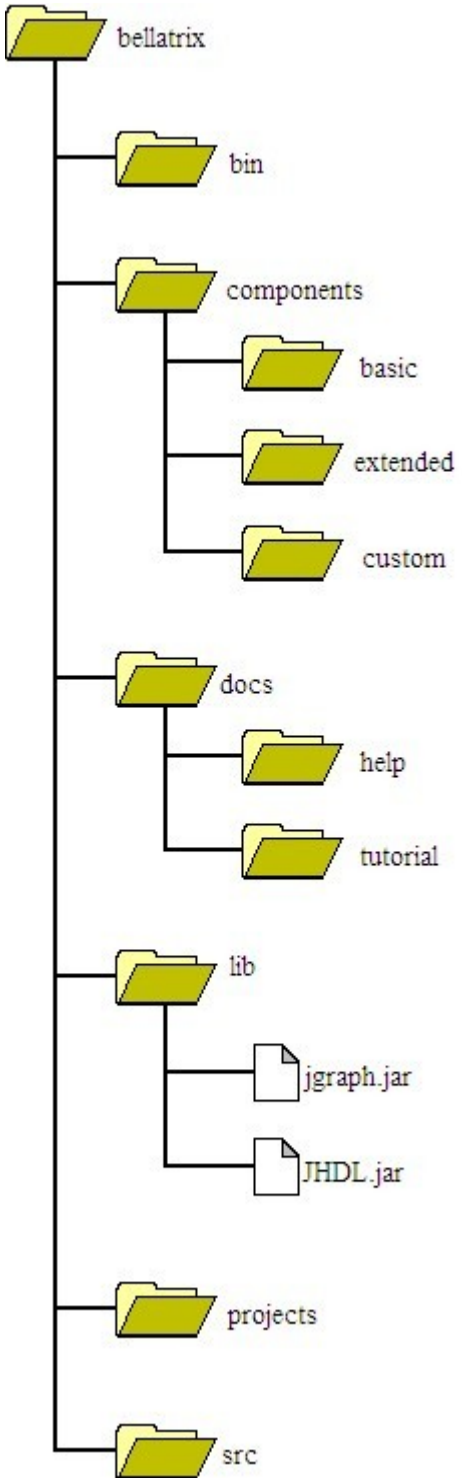


Figure 29: Directory structure of Bellatrix

This figure shows the directory structure of Bellatrix. The top-level directory will have the application icon, splash screen, base configuration and shortcut files, that are mostly standard in all applications.

The executable files of the application will be kept under the `bin` directory. All the digital circuit components will be under components directory. This directory will have three subdirectories: `basic, extended` and `custom`, for basic, extended and custom components respectively. These directories will keep the components' configuration, image and behavior in separate files under the components' unique names.

Under the `docs` directory, the user will find the help and tutorial of Bellatrix. These files will be also accessible from the help menu of the application.

The `lib` directory will keep the specific libraries that will be used in the application. Two of these are JHDL.jar and jgraph.jar. JHDL is the simulation library that we explained in detail in previous work reports. Jgraph is a drawing library for Java, which uses native Java swing and Java2D libraries.

The `projects` directory will be a default directory for user projects to be saved. The user, by the way, will be able to save his/her projects under any other directory.

The `src` directory is an optional directory. If the user wants to work with the source code of Bellatrix, he/she will be able to install the source code with the installation of Bellatrix.

5.3 File Formats

`{component}.dat` : Save format of custom components. The custom components will be kept in serialized object form, others will be precompiled objects.

`{component}.inf` : The configuration of the component. This configuration file keeps the input/output ports of the component. These ports are kept in relative coordinates to the component image. This configuration is required while drawing the component. The input/output ports will be the anchor positions in the drawing. No lines will be able to be connected to any other points on the drawn component.

`{component}.jpg` : The image file of the component. Custom components' images will be renamed to the component's name and copied under the directory where the configuration file is saved.

`savefile.bx` : The saved project format. The save file will be just the serialized form of the CircuitEngine object.

`savefile.lgf` : The Diglog file format. Bellatrix will be able to save projects in Diglog file format.

5.4 Threads

Bellatrix has two threads. They are as follows:

- Draw
- Simulate

These classes implement the Runnable interface and can be run concurrently. They will call wait(), notify() methods of Runnable interface to interact with each other and to perform the concurrency in a proper way.

Bellatrix has both a continuous simulation (on the fly, always running) and a one step simulation support. In the continuous simulation mode (which is the default case unless changed from the GUI), the circuit will be simulated again each time the user makes a change in its design which requires synchronization between simulate() method of Simulate class, and all the methods of the Draw class that are modifying its lineCellVec and compCellVec vectors which are namely drawLine, drawComponent, removeLine and removeComponent methods. Also note that since Bellatrix makes a continuous simulation, Draw thread has higher priority then the Simulate thread which enables performing the simulation always according to the the latest version of the circuit.

In order to handle the synchronization, for each modify-simulate cycle, simulate method must wait for any method that modifies the lineCellVec and compCellVec vectors of Draw to finish its execution. For example if a method is modifying the vectors in the Draw thread, and the simulate method is executed in the Simulate thread at the same time, the Simulate thread must wait for the Draw thread. When the method finishes its job, it will notify the Simulate thread to continue from the point it left its execution.

Also if the user initiates a change in the drawing of the circuit, the Draw thread takes over the CPU in order to modify the lineCellVec and compCellVec vectors accordingly since Draw thread has higher priority than the Simulate thread.

6 Dynamic View

6.1 Action Specification

Actions are data messages providing the interactions among the users in the system. The actions are transferred through action channel. The action types are defined in the following format:

6.2 Action Types

Action Generator: is the user who creates the action.

Action Event: is the event that triggers the action

Action Data: keeps the required data to perform the action. First element of the Action Data String keeps the action type which defines which event done by the event generator, i.e. action type tells us the user selected a component from the component panel. And the remaining part of this string defines the action arguments. The number of arguments can change according to the type of the action.

Action Location : describes where the action event is triggered.

Action Processes: describes the processes to be performed when the action is triggered.

6.2.1 Select a Component From Workspace

Action Generator: User

Action Event: Clicking on a component in the workspace view

Action Data: <Action type> <Component type>

Action Location: GUI Workspace Pane

Action Processes: The component that is clicked from the related workspace panel is selected.

6.2.2 Select a Component From Drawing Area

Action Generator: User

Action Event: Clicking on a component in the drawing area

Action Data: <Action type> <Component type>

Action Location: GUI Drawing Area

Action Processes: The component that is clicked with the mouse in the drawing area is selected.

6.2.3 Add a Component

Action Generator: User

Action Event: Clicking on a component in the workspace view

Action Data: <Action type> <Component type>

Action Location: GUI Drawing Area

Action Processes: The selected component from the workspace pane, is dragged to the drawing area and is drawn to the location pointed by the cursor.

6.2.4 Delete a Component from Pop Up Menu

Action Generator: User

Action Event: Choosing the Delete Option from the Pop Up Menu for the selected component.

Action Data: <Action type>

Action Location: Menu Bar

Action Processes: The selected component is deleted from the drawing area

6.2.5 Move a Component

Action Generator: User

Action Event: Dragging a component in the drawing area.

Action Data: <Action type>

Action Location: GUI Drawing Area

Action Processes: The selected component is moved in the drawing area by dragging the mouse.

6.2.6 Draw a Wire

Action Generator: User

Action Event: Dragging the mouse by pressing the left button continuously in the drawing area.

Action Data: <Action type> <Drawing Coordinate>

Action Location: Drawing Area

Action Processes: The wire is drawn in the drawing area between start and end points. The start point of the line is specified by the location pointed by the cursor when the left mouse is first pressed and the end point is specified by the location pointed by the cursor when the left mouse is finally released.

6.2.7 Select a Wire

Action Generator: User

Action Event: Clicking on a wire in the drawing area

Action Data: <Action type> <Component type>

Action Location: GUI Drawing Area

Action Processes: The wire that is clicked with the mouse in the drawing area is selected.

6.2.8 Delete a Wire from Pop Up Menu

Action Generator: User

Action Event: Choosing the Delete Option from the Pop Up Menu for the selected component.

Action Data: <Action type>

Action Location: Menu Bar

Action Processes: The selected component is deleted from the drawing area

6.2.9 Move a Component

Action Generator: User

Action Event: Dragging a wire in the drawing area.

Action Data: <Action type>

Action Location: GUI Drawing Area

Action Processes: The selected wire is moved in the drawing area by dragging the mouse.

6.2.10 New Project

Action Creator: User

Action Event : Choosing the New Option from the Project Menu in the Menu Bar

Action Data: <Action type>

Action Location: Menu Bar

Action Processes: An empty project is created.

6.2.11 Open Project

Action Creator: User

Action Event : Choosing the Open Option from the Project Menu in the Menu Bar

Action Data: <Action type> <Project Name>

Action Location: Menu Bar

Action Processes: The specified project is opened.

6.2.12 Save Project

Action Creator: User

Action Event : Choosing the Save Option from the Project Menu in the Menu Bar

Action Data: <Action type>

Action Location: Menu Bar

Action Processes: The current project is saved.

6.2.13 Print Document

Action Creator: User

Action Event : Choosing the Print Option from the Project Menu in the Menu Bar

Action Data: <Action type>

Action Location: Menu Bar

Action Processes: The selected document is printed.

6.2.14 Exit Bellatrix

Action Creator: User

Action Event : Choosing the Exit Option from the Project Menu in the Menu Bar

Action Data: <Action type>

Action Location: Menu Bar

Action Processes: Exits from the program.

6.2.15 Undo the Last Action

Action Generator: User

Action Event: Choosing the Undo Option from the Edit Menu in the Menu Bar

Action Data: <Action type>

Action Location: Menu Bar

Action Processes: The last action performed in the drawing area is undone.

6.2.16 Redo the Last Action

Action Generator: User

Action Event: Choosing the Redo Option from the Edit Menu in the Menu Bar

Action Data: <Action type>

Action Location: Menu Bar

Action Processes:The last action performed in the drawing area is redone.

6.2.17 Add Sheet

Action Generator: User

Action Event: Choosing the Add Sheet Option from the View Menu in the Menu Bar

Action Data: <Action type>

Action Location: Menu Bar

Action Processes: A blank sheet will be added to the current project.

6.2.18 Zoom In/Out

Action Generator: User

Action Event: Choosing the Zoom Options (%50,%100) from the View Menu in the Menu Bar

Action Data: <Action type>

Action Location: Menu Bar

Action Processes: Drawing Area will be zoomed in/out.

6.2.19 Show Status Bar Mode

Action Generator: User

Action Event : Choosing the Status Bar Option from the View Menu in the Menu Bar

Action String: <Action type>

Action Location: Menu Bar

Action Processes: The current status bar mode will be displayed in the Status Bar

6.2.20 Find Custom Component

Action Creator: User

Action Event : Choosing the Find Component Option from the Component Menu in the Menu Bar

Action Data: <Action type> <Component name>

Action Location: Menu Bar

Action Processes: The search is performed for the specified component name.

6.2.21 Save Custom Component

Action Creator: User

Action Event : Choosing the Create Component Option from the Component Menu in the Menu Bar

Action Data: <Action type> <File name>

Action Location: Menu Bar

Action Processes: The specified component is saved as a custom component in the specified file name.

6.2.22 Run Simulation

Action Generator: User

Action Event : Choosing the Run Option from the Simulation Menu in the Menu Bar

Action Data: <Action type>

Action Location: Menu Bar

Action Processes: The simulation of the circuit is started.

6.2.23 Pause Simulation

Action Generator: User

Action Event : Choosing the Pause Option from the Simulation Menu in the Menu Bar

Action Data: <Action type>

Action Location: Menu Bar

Action Processes: The simulation of the circuit is paused

6.2.24 Stop Simulation

Action Generator: User

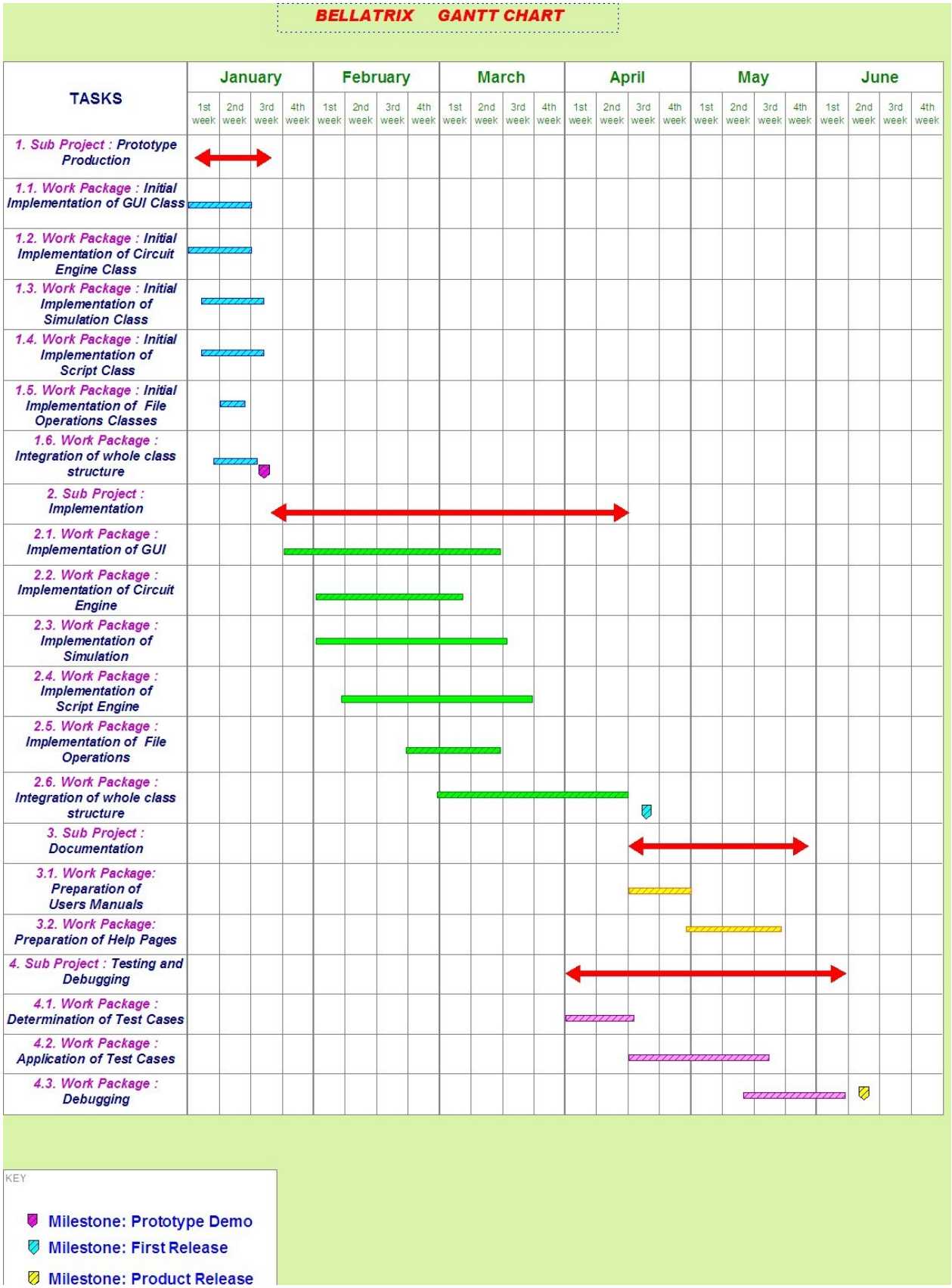
Action Event : Choosing the Stop Option from the Simulation Menu in the Menu Bar

Action Data: <Action type>

Action Location: Menu Bar

Action Processes: The simulation of the circuit is stopped.

7 Project Plan



8 References

[1] Java Hardware Description Language, JHDL Home Page, www.jhdl.org
[2] Jgraph Home Page, www.jgraph.com
[3] Sun Java Home Page, http://java.sun.com
[4] Python Scripting Language Home Page, www.python.org
[5] Jython, The Java Port of Python Language, www.jython.org