Ruprecht-Karls Universität Heidelberg
Institute of Computer Science
Research Group Parallel and Distributed Systems

Internship

Analysis and Extension of the JavaGUI for PIOsimHD

Name: Samantha Dulip Withanage
Matriculation number: 2371809
Supervisors: Prof. Dr. Thomas Ludwig, Julian Kunkel
Date of submission: 10. Februar 2009

1 Abstract

This document describes the graphical user interface for the Parallel I/O Simulator(PIOsim) for clusters, developed by the Parallel and Distributed Systems Research Group of the Ruprecht-Karls Universität in Heidelberg, Germany. The graphical user interface is based on a previous implementation. [Bra].

The PIOsimGUI is a swing and awt based java application. The underlying modeling architecture Piosim which is also a java application was completely rewritten by Julian Kunkel during the internship period to facilitate the complex requirements of modeling the cluster. The integration of the model to the existing graphical user interface was executed by the author. New programming concepts were added to the programming model and to the graphical user interface in order to provide efficient and dynamical graphics rendering. The existing GUI was extended to facilitate more than one switch, component hosting machines that hold a set of components. A basic point-to-point connection drawing concept was introduced for connecting the cluster component. An easily modifiable attribute setting concept was added and the programming was executed in a one-to-one mapping of the model object for easily implementation of newer component types. Some of the restrictions of the earlier drawing area were eased as well.

The document also discusses the technical difficulties that the author confronted in extending the graphical user interface. Most of the restrictions that were cased by the underlying swing and awt libraries are discussed and illustrated in examples.

Furthermore, the document describes the concepts of the Model-View-Controller (MVC) architecture that is a generally accepted and mostly used concept of user interface designing. The relation of the current model to the MVC architecture is also discussed.

The Eclipse Standard Widget Toolkit (SWT) [A07] is anaylsed in the last chapter. A sample SWT paint application is included as an executable jar file which comes with the eclipse SWT installation.

Contents

1 Abstract **2**

2 Introduction **4**
2.1 Methodology 4

3 Use cases **5**

4 Model-View-Controller Concept **16**
4.1 Introduction 16
4.2 Comparison Current Model vs. MVC 17

5 Current Model **18**
5.1 Model 18
5.2 View 20
5.3 Functions 21
5.4 JPanel restrictions 23

6 SWT vs. AWT **25**
6.1 SWT Introdcution 25
6.2 The line drawing concept of SWT vs. AWT 26

7 Examples of better design **27**

8 Conclusion **28**

Bibliography **29**

2 Introduction

The parallel I/O Simulator (PIOsim) is a model - driven graphical user interface designed for simulating parallel programs running on clusters. A cluster in the context of the document is understood as a set of clients, servers and switches that are connected with network cables to act as one-logical unit, which executes tasks in parallel. The graphical user interface prototype that was developed by Matthias Braun and was later extended by Julian Kunkel is the basis of the user interface.

The main purpose of the internship is to develop a consistent model and to extend the limited functionalities that exist in the current GUI. Another important task is to decouple the components of the model and the GUI into separate packages for a clear separation and for the facilitation of autonomous development. As the model concept that was adopted earlier was not necessarily fulfilling the complex requirements of simulating a cluster, Julian Kunkel designed a completely different model than that was adapted earlier. The graphical components, that was existing earlier was integrated into the model and necessary changes in the model for the integration was executed during the internship period. The following sections describe the model package and the GUI packages that currently exist.

2.1 Methodology

The following methodology is used in this document.

1. PIOsim is the abbreviation for the Parallel Input/Output Simulator.

2. GUI is the graphical user interface of the PIOsim. It is also referred as user interface or the user interface.

3. Component, Graphical component is the word used for the visual object of the component of a cluster such as servers or clients.

4. Panel is defined as a graphical user component which holds a set of other components. E.g. the graphical user in label [1] of the figure 5.3 shows a panel component.

5. Drawing Area is a canvas where the components are drawn.

6. The term [DE], when used before any word is an indication that it is a word in the German language or a key of the German keyboard. DE is the abbreviation for Deutsch.

3 Use cases

In this chapter, a real world cluster is introduced and the tasks that have to be executed when modeling the cluster is also described. Some of the usability features were decided after observing some of the fascinating open source programs such as NetBeans [A10] and the Eclipse Software Framework [A11]. Another most considered fact is the type of users that would use the tool. The users who will use the tool will be mostly linux users who are more used to the key board combinations than moving the mouse. Therefore mostly user key combinations for executing were also considered when defining use cases. Some of the possible implementation suggestions were also added in the user case diagrams. Please note that the user cases are colored in fading green and the possible solutions are colored in the fading yellow.

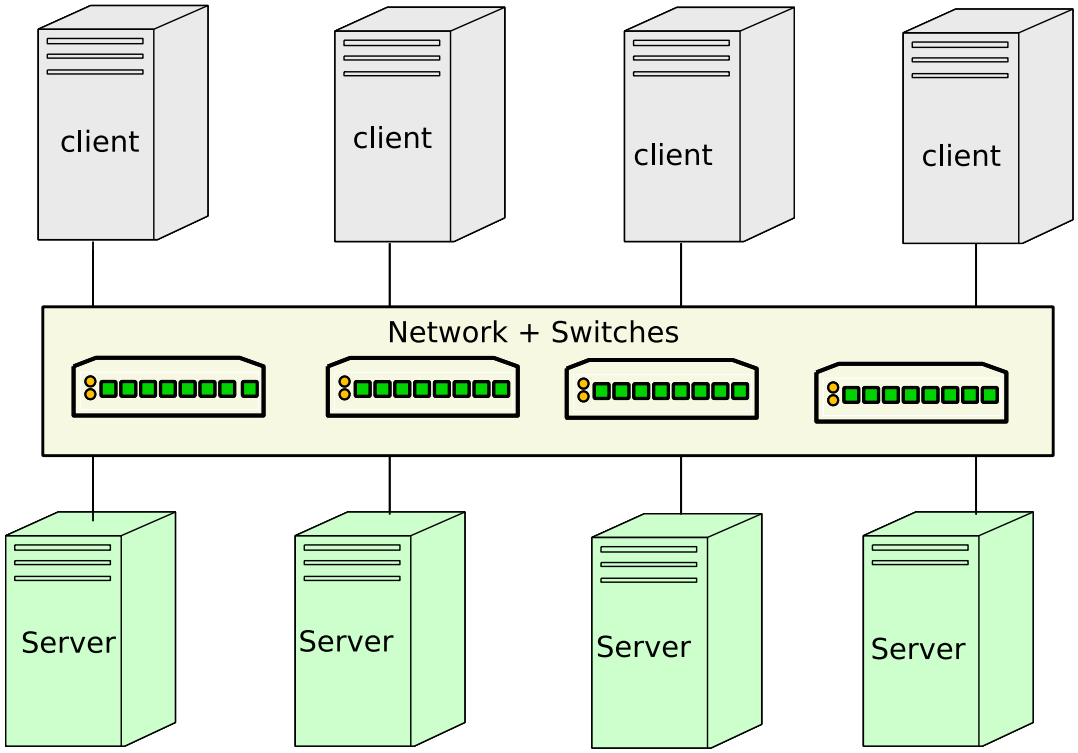

Figure 3.1: Cluster Architecture

The architecture of a real-world scientific cluster that is calculating parallel programs appears similar to the figure 3.1. The clients are computers where the parallel programs are running. The clients execute special programs that can do tasks in parallel. In a homogeneous cluster every client has the identical hardware. But there are also heterogeneous clusters where the hardware configurations of the clients differ. The discussion will be focused explicitly on the homogeneous clusters.

In the cluster, there are servers which access the data, that is being read or written by the parallel programs that are running on the clients. Servers posses disk I/O systems which hold the data.

Clients and servers are connected to a network with network cards and network cables. The network may not be typical Ethernet, but other high performance networks such as myrinet. Inside the network, there are switches that transfer the network data.

The main user cases of the visual modeling structure are listed underneath and is depicted in the figure 3.2 .

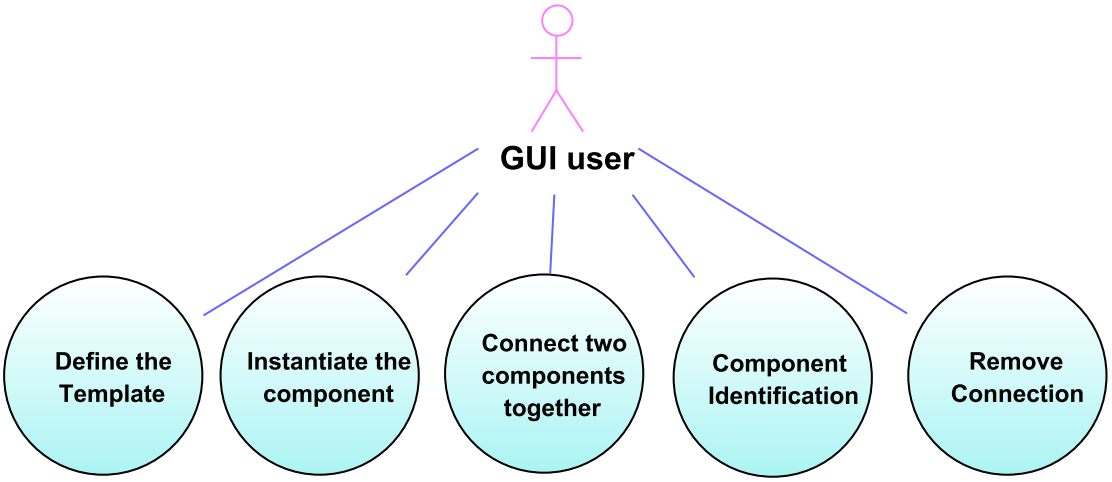

Figure 3.2: Basic user cases

1. **Define and generate the templates.**
   In a homogeneous cluster, the hardware of the computer nodes are identical. Therefore it is necessary to define a template architecture to facilitate defining similar hardware components which only defer from there label (A name or a unique id). There are also global settings in the cluster, which set using the template structure.

2. **Instantiate the component.**
   A cluster component has its own attributes. For each component, the attributes has to be set either by setting them individually or by modification of a template. Changes to the templates should be propagated efficiently, for instance this allows rapid testing of the cluster environment with different network speeds.

3. **Identify between connectable or unconnectable components and network cables.**
   The components are categorized in to three groups according to their connectability to external components. The first type is the unconnectable component that cannot be connected to a network directly, rather holds the components that interacts with the

network. E.g for unconnectable components are clients or servers that are equipped with network cards. The second type of component is the network component such as Network interface cards which are connected to the network cables. The third type is the connection cables. They are distinguished from other network components because they have on either end a network component attached to it, unlike the network components.

4. **Connect two components together.**
   After components are set, their Network components must be made connectable. Explicitly, this is achieved by setting a network cable between two connectable components.

5. **Component identification.**
   A mechanism must be in place to identify the component type and their attributes.

6. **Remove a connections.** Disconnecting a component is identified as removing the cable that connects the component to the cluster.

Following use cases are defined not only for the basic functionalities but also for the better user experience in the GUI.

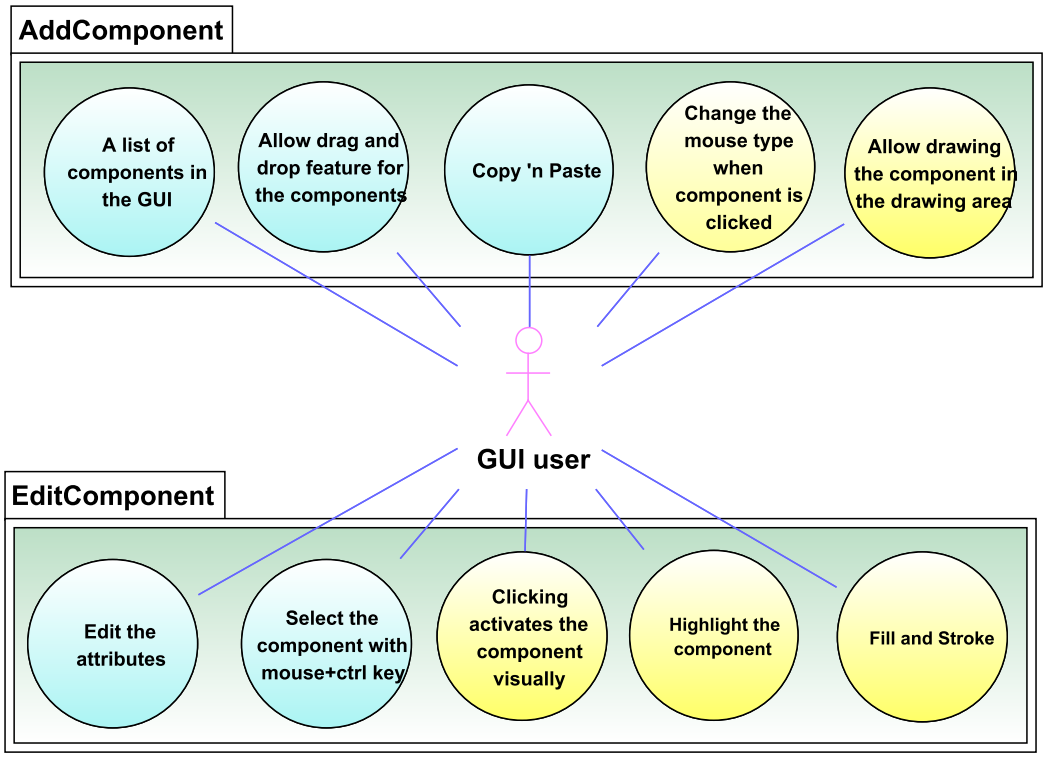

Figure 3.3: Adding and editing components in the GUI

1. **Adding a new component to the cluster.**

- **A list of components in the GUI**
  A list of component types has to be defined in a panel for the gui user to select them for editing. The panel must hold all the components that are available in the real-world cluster.

- **Allow drawing the component in the drawing area.**
  If the component can be drawn using a pre-defined model, it will enhance the user experience. User may wish to draw the component by enlarging or minimizing a standard model picture.

- **Copy and Paste**
  Duplicating the existing cluster components is necessary when constructing the cluster components or adding components that are similar to the already existing components.

- **Change the mouse type when component is clicked.**
  When the component is clicked it is desirable that the mouse type is changed indicating that component adding is activated. This effect helps the user to know that the gui is in an editable modus.

- **Allow drag and drop feature for the components**
  For the user to create a direct projection of the real-world cluster in the GUI, it may be needful to relocate the components due to the infrastructure changes.

The figure 3.4 in the Netbeans shows this effect, when editing the uml diagrams. Figure 3.4 shows that an actor is chosen and highlighted. The gui user can insert an actor component in to the drawing area. The component has to be draggable inside the drawing area to facilitate repositioning. Another interesting feature may be to cut and paste the components. As most of the moderner graphical applications allow, copying and pasting the component using either by hot keys or using a list by right clicking is also desirable.

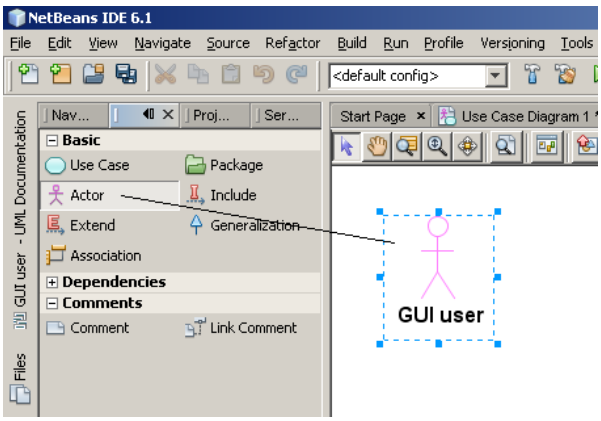

Figure 3.4: Add component Extensions

2. **Editing the components in the cluster.**
   Use cases for editing the component are summarized in the figure 3.3. Editing the component and setting the attributes has to be one of the most user friendly and optimized

interactions, because this action has to be repeated a lot when setting a cluster component.

- **Edit the attributes**
  Editing the attributes directly in the GUI is compulsory for setting the attributes of the cluster components. This may be implemented by allowing the user to directly edit the attribute after clicking on the component and/or on the respective component panel.

- **Select the component with mouse+ctrl key**
  Users need to select the components of the cluster for various purposes. E.g a user may wish to run a program in a set of components and want to click and select them. A generally used procedure is pressing the ctrl key and clicking on the desired component. For the latter, the attribute panel is automatically activated, simultaneously when the component is clicked. The user has to have the ability to edit the component attributes directly there. This effect is implemented in the current GUI (see figure 3.5).

- **Clicking activates the component visually**
  The color of the clicked component may be changed or the borders of the component may be highlighted.

- **Highlight the component**
  When the component is clicked there must be a visual indication that the user is editing the selected component.

- **Fill and Stroke**
  Fill color and the stroke colors can be customized for visual identification of different types of components. An example use case scenario can be a cluster that has two types of processors in it's clients. E.g the clients with intel processor may be filled with color blue and the AMD clients with green.

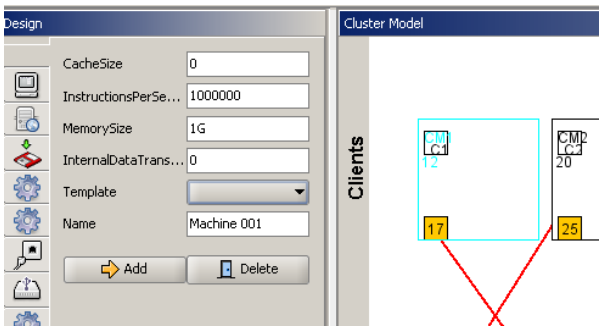

Figure 3.5: Edit component attributes

3. **Remove a component.**
   The removing of a component can be put in to the following categories as depicted in the figure 3.6. The tasks for removing components are defined with the background idea of how the model components are changed, when the real-world cluster is relocated, it's components are changed. E.g a very possible scenario is that after a period of time, the

network is upgraded to a more faster version. The GUI must allow mechanisms to remove components efficiently.

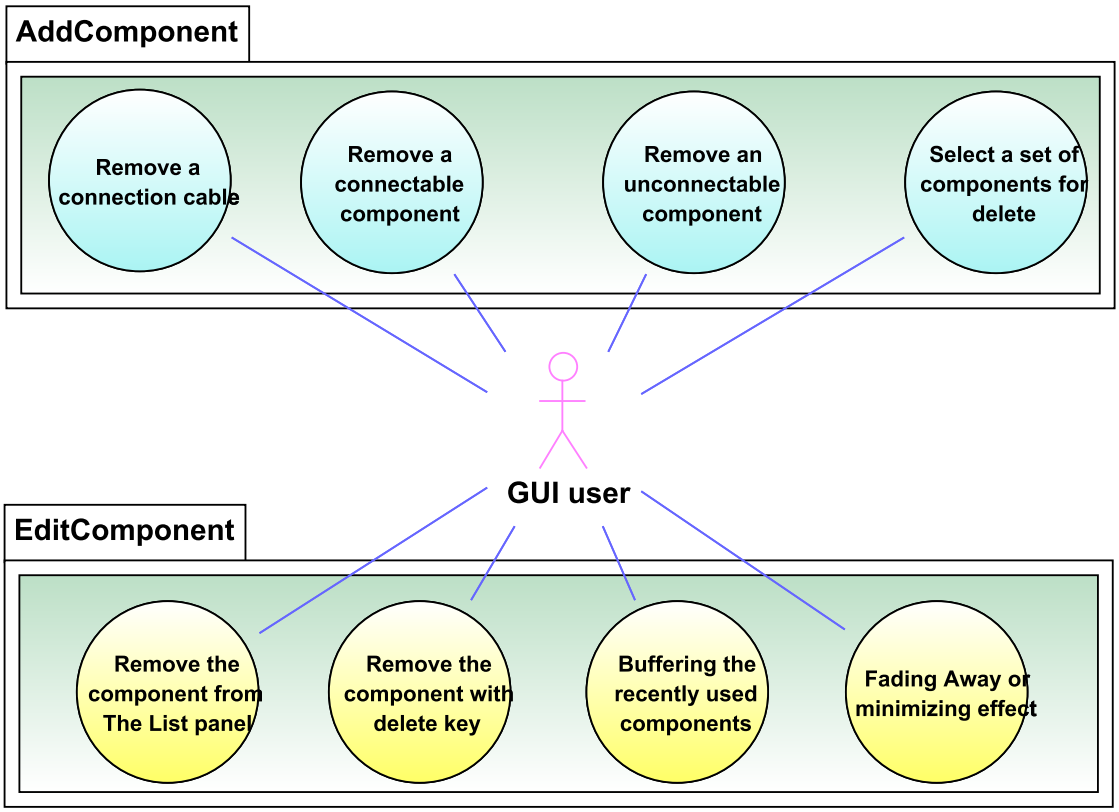

Figure 3.6: Remove Components

- **Remove a connection cable**
  In this case only the connection component is removed from the model.

- **Remove a connectable component**
  In this case, the connecting component has to be removed and the connecting cable has to be removed as well. E.g Removing a Network Interface card (NIC) automatically triggers removing the connecting cable from the model.

- **Remove an unconnectable component**
  Removing a component such as server triggers removing all the sub components such as disks and NICs inside it and the connection cable, if there is a connection to the network.

- **Select a set of components for delete**
  Another important feature is selecting the components in the drawing area with the tab key. This is an important feature to some of the users, who are used to press the tab key. Component selection may be activated from the following priority as in a regular web browser. The tab key begins from the top left hand corner. It

moves in a row from left to right. When finished, it moves to the row which lies below. The coordinates for the movement can be extracted by using the top-left corner coordinates of the component.
A second movement strategy is to prioritize the components according to pre-defined conditions. E.g When a component that is holding sub-components is selected, the movement may first in all the sub-components before moving to the component which lies to it's right.
Selecting a set of components for deleting must be executed with the help of a convenient method. A mostly used way is to click on the intended components while pressing the ctrl ([DE] Strg) key. Select can also be used to export a set of components to another program such as a graphic design program. Please refer to the use case diagram 3.7.

- **Remove the component from The List panel**
  As we saw in the edit section, the users can activate the attributes panel to edit the attributes. A delete button is desirable to remove the component inside this panel. This feature is already implemented in the current gui.

- **Remove the component with delete key**
  When a component is selected, it is a general feature that it can be removed with the delete key ([DE]Entf Taste).

- **Save a recently removed components list in a buffer** for further reference. This feature can be based on the current gui session. This may be helpful when the users are removing a component or a set of components from the cluster for hardware errors. The GUI user can save them in a list and later add them when they are later serviceable.

- **Fading Away or minimizing effect**
  A visual effect for removing the component may be to show a minimizing rectangle or a fading away effect.

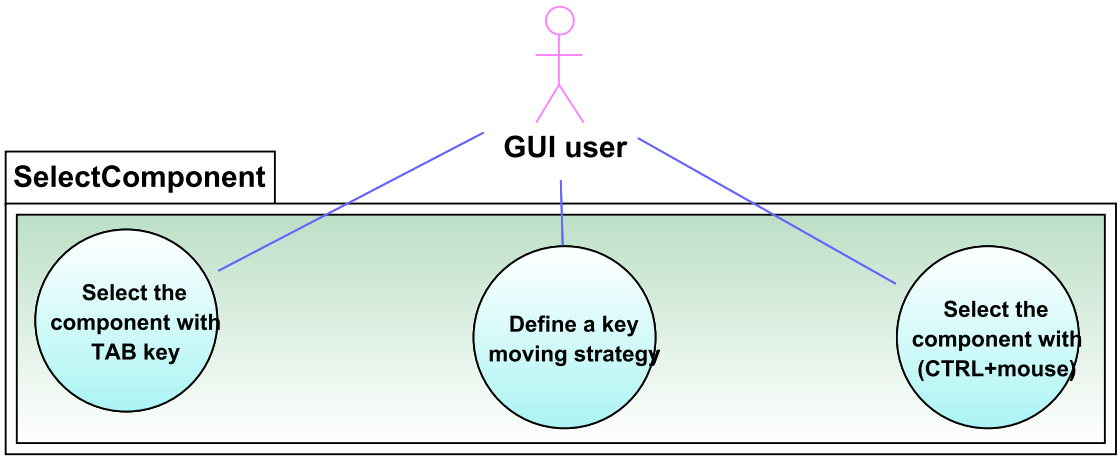

Figure 3.7: Selecting the component

4. **Tasks menu for the right mouse click**
   The Menu that appears by the right mouse click on a component allows all the tasks that have to be executed on the component. Please note that some of the ideas are inspired from the Adobe Fireworks [A12], a graphics design program. The Right Click Menu actions are listed not in a priority manner to allow the designer free imagination. Figure 3.8 shows a typical right mouse click menu.

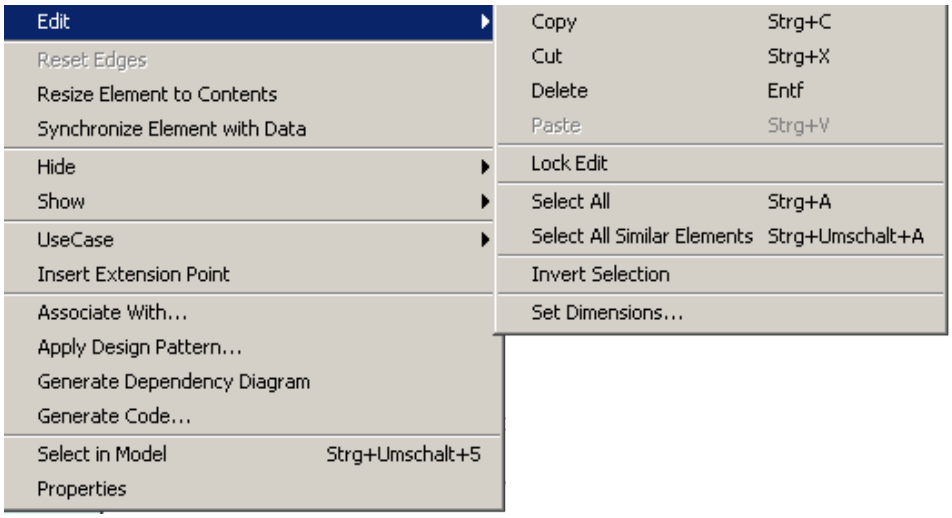

Figure 3.8: A sample right mouse menu from NetBeans

All the use cases for the right mouse click menu are listed in the figure 3.9.

- **Copy :** generate an identical copy of the component. To generate the identical copy user has to get all the attributes of the component, it's position and other visual features that are added to the component.

- **cut :** The same as copy, but the original component is removed from the drawing area. The new position can be calculated using an pre-defined algorithm. A desirable feature is also to hold the component in a well-accepted format for graphical programs. Then the users can add the component in to a text document or graphic designing application.

- **Paste :** After a component is copied or cut, users can paste them to a pre-defined position or to the position direct under the mouse.

- **Duplicate :** This function can be the same as copy, but it will allow the user to make an exact copy of the component. The graphics designer may able to customize the Copy and Duplicate to accommodate more user friendly functions.

- **Delete :** This allows deleting the component.

- **Undo :** If a user has accidently deleted a component, this function will allow restoring the component.

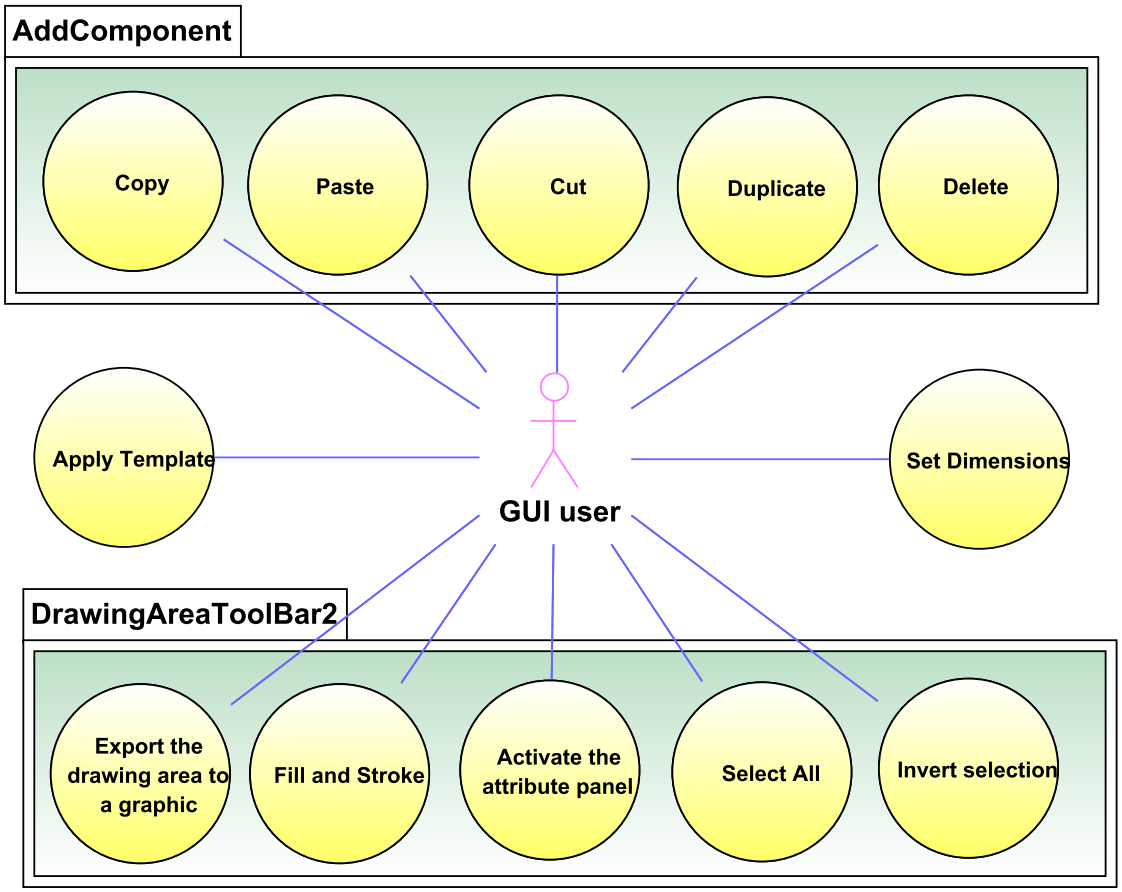

Figure 3.9: Use cases for Right Mouse Click Menu

- **Fill and Stroke :** Fill color and the line color of the component may be changeable. This will allow the user to customize the components. Fill colors may be an inspiration for the copy and duplicate function. E.g. the Copy function may only copy a standard color component, but the duplicate will also copy the customized colors of the component.

- **Select All :** This functions selects all the components in the drawing area. It may be necessary for exporting the components to another program.

- **Invert selection:** This function is not a familiar function, but may be necessary to select a large set of components avoiding a few selected ones.

- **Select all similar components:** May be helpful to select a set of components from the same type. A real-world use case scenario may be selecting all the network interface cards and setting their latency attribute from a new template.

- **Set Dimensions :** Most of the gui users may not want to move the mouse to generate components in a customized size for standardization. Therefore setting the width and height attributes from a window may reduce much of the time of the gui

user.

- **Apply Template :** A simple way to set the template is to integrate the temple setting in the right mouse click menu. This will allow the user to select a set of components using the mouse and set the same template for all of them.

5. **The drawing area tool bar**
   A tool bar for the drawing area may help the gui user to have a better control of the GUI.

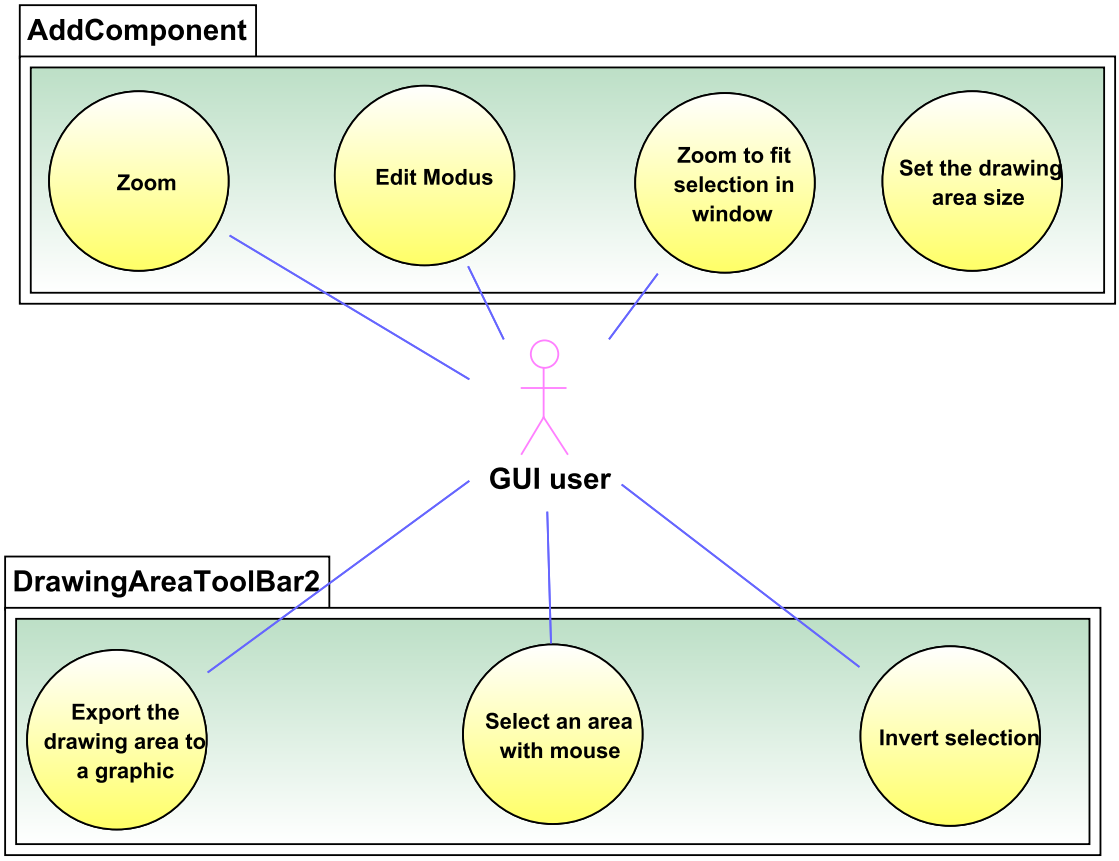

Figure 3.10: Tool bar for the drawing Area

- **scrollable drawing area :** The drawing area shall expand when the number of components increase and the space they are taking increases.

- **Zoom :** A zoom function shall help to focus in and out of the components in the drawing area. This function is required when modeling large clusters. User may want to edit a particular sector of the cluster and want to zoom into the respective component. A precondition for the zoom function is the use of vector graphics for the components.

- **Edit modus:** The modus of the drawing area should be made uneditable. When a test is running on the cluster, the user may only want to watch how the system

behaves and not to accidently edit the component attributes when an application is running. E.g the drawing area may be made automatically uneditable during the test execution.

- **Select an Area with mouse :** Selecting an area with the mouse is helpful for a lot of tasks. Some of them are listed follow.

- **Set the drawing Area size:** It may be an added advantage if the user could set the drawing area to a pre-determined size. E.g setting the area to DIN A4 size would help, for optimized printing of a cluster environmet.

- **Zoom to fit selection in window :** Zooming the selected section in to the full visible area may be helpful to easily edit a section of the cluster.

- **Export the drawing area to a graphical format :** Formats such as png ,jpg or svg can be used to export the drawing area to a graphics file.

- **Naming the drawing area:** Customized name adding to the drawing area may be needed in order to mark a cluster name.

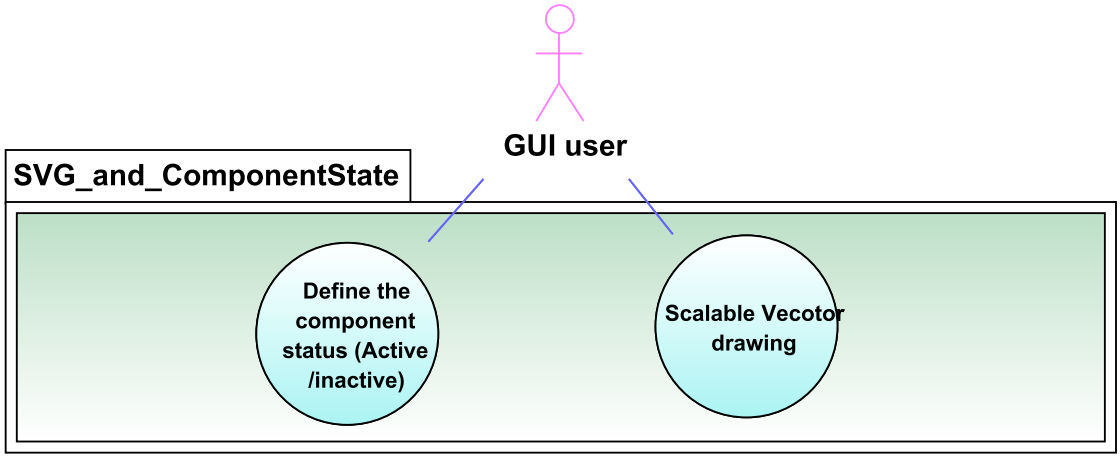

Figure 3.11: Set the component state and use scalable vector graphics

6. **Define the state of the component**

   - **inactive :** When modeling a particular parallel program, not all the components are necessarily required for it. Therefore the user may want to see, what components are active for the certain test.

7. **Vector graphics :** Vector graphics allows scalable components without distorting the quality, when they are enlarged or when they are minimized. If the component drawing is planed with automatic scalable methods, it would be helpful when exporting scalable vector graphics (SVG).

4 Model-View-Controller Concept

4.1 Introduction

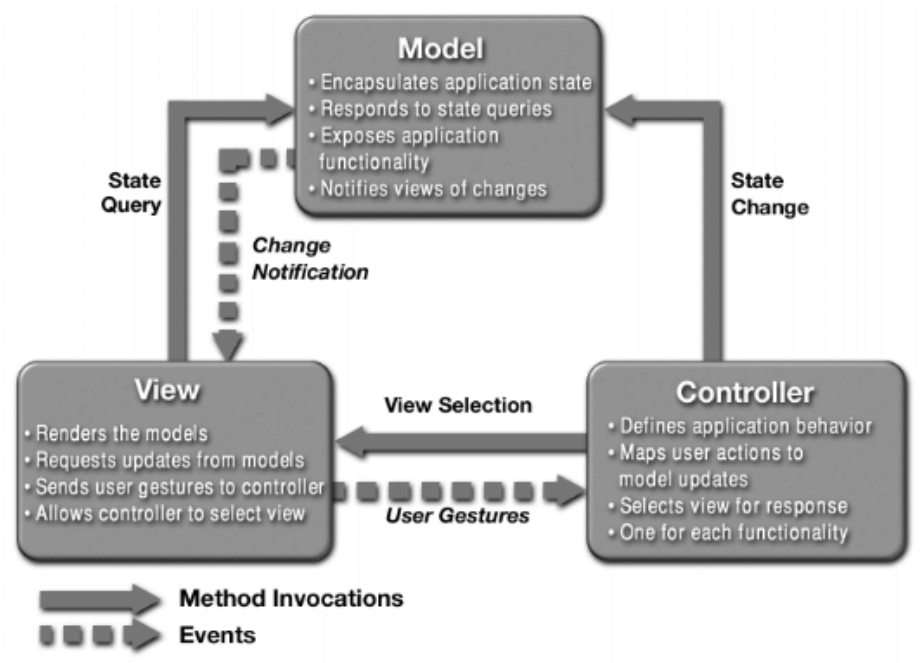

Figure 4.1: Model-View-Controller Architecture [Mic]

Figure 4.1 from the sun's blueprints shows the graphical representation of the model-view-controller concept used in Swing. Model-View-Controller is a structural architecture model often used for designing graphical user interfaces. The model represents the logical representation of the data or the real-world model approximated to the software. The View is the representation of the content in the model in the graphical user interface. The controller defines how the user interaction is defined in the model. Controller forwards the actions performed on the view in to the model [Ins].

There are two approaches to connect the model and the view, namely the push-model and the pull-model. In the push-model, the view registers on the model in order to get the notifications. Whenever there is a change in the model, it notifies the view and view gets updated. In a pull-model, there is a mechanism in the view where it calls the model to get an updated version of the model.

4.2 Comparison Current Model vs. MVC

The current design of the PIOsim GUI is in many ways similar to the MVC concept. As explained in the current model chapter (see chapter 5) the isolation of the model from the GUI depicts the characteristics of the Model - View separation that shall be necessary in a MVC architecture. The controller which triggers the events is integrated to the view. Each graphical component implements the MouseListner() of the **java.awt.event** package. In this way, the current implementation of the GUI acts as a pull-model, in which the GUI is regenerated only when the view inquires the model for an update.

Although the MVC is implemented in the traditional sense, the current GUI has some draw backs.

One of the most significant of these is the algorithm-based component position assignment. The current strategy of determining the position of a component works as following. The JDrawingArea is separated in to three virtual horizontal sections. The top-most section is reserved for the maschines that have only clients inside them. The middle section is reserved for the switches. The bottom section is reserved for the maschines that posses at least one server and clients.

The algorithm for the position calculation first identifies the maschines and switches. Then it checks whether the maschines are having at least one server inside it. According to the above criteria they are put in the user interface from left to right. If any of the component groups exceeds the visible GUI area, a scroll bar is generated automatically to facilitate their placement.

This positioning system has the disadvantage of not saving the order of the components which in return restrict the natural order of the real-world cluster components. A possible use case is that the user map the cluster to the GUI in the same way, that it is located in physical form. This implies the need for saving the position of the cluster component permanently. This can be in form of extending the current XML Model file with additional attributes or using an additional XML file to save the positions. The author suggests to user the latter idea, due to it's association with the MVC Concept, thus making the view data separate from the model data.

5 Current Model

5.1 Model

All the model related components are included in the package **de.hd.pvs.piosim**. The model package is located in the CVS Host : p**vs-cluster.informatik.uni-heidelberg.de** under the repository module **piosim-shared**. The **de.hd.pvs.piosim** package is depicted in the figure 5.1.

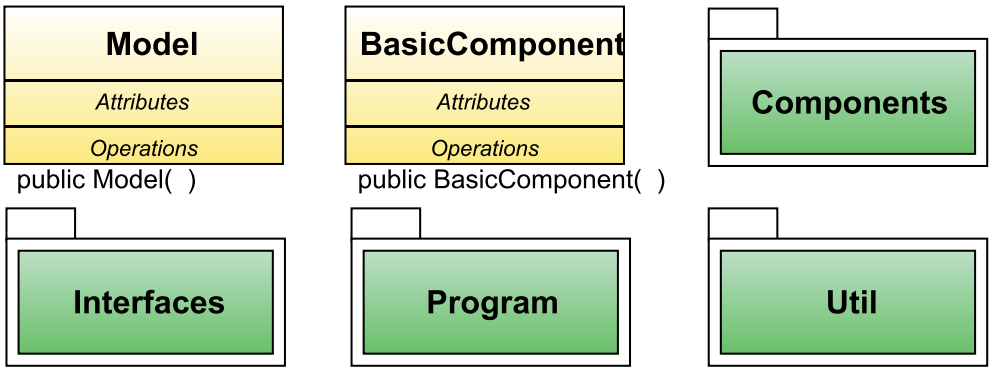

Figure 5.1: Main packages of the model **de.hd.pvs.piosim**

Class **BasicComponent** represents the basic characteristics of a component in the cluster. The Class **Model** represents the complete structure and their relations and is accessed by the graphical user interface to visualize the model. The package **components** contain all the components that are included in a cluster. They all are sub classes of the class **BasicComponent**. At present, the package **interfaces** contains the classes that are required to interface the model with the XML files. The classes that are related to the parallel programs such as commands, distributions and MPI-related data are stored in the **program** package. The package **util** contains the helper classes such as time functions or file read/write functions.

As shown in figure 5.2 the UML Diagram shows the current model that is used to generate the graphical view of the computer cluster. As shown in the attributes part in the UML diagram, the model consists of components lists that map the real world cluster components. A component called maschine was defined to host a set of clients and servers. Switches and maschines are connected through network connections. The model also consists of a template

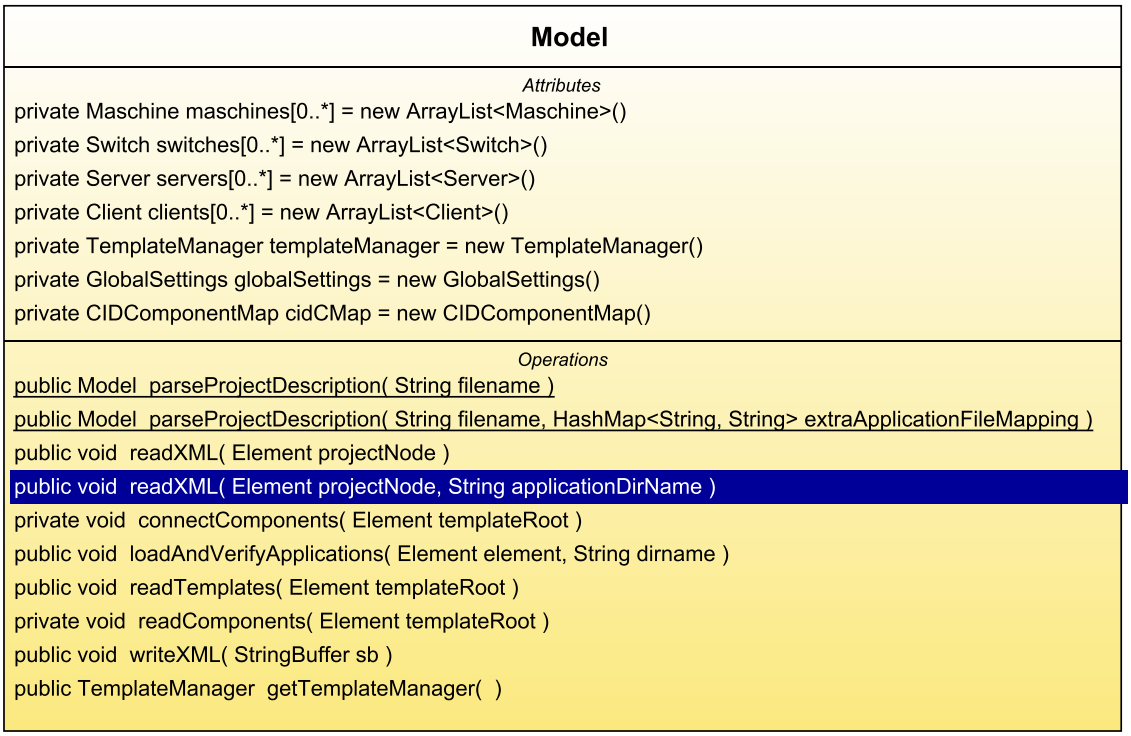

Figure 5.2: UML Diagram of Model.java

manager that can set template based values to all the components in the cluster. The GlobalSettings class is responsible for setting the common attributes that go beyond the boundaries of a single component. The cidMap (Component ID map) is a HashMap that is responsible for holding unique ids assigned to the respective components.

All the components are saved in an XML file and accessed by the model to generate the classes. The java reflection class is used to generate the Objects from the XML data. From the available methods following methods are significant to the function of the model class.

- **readComponents()** read the components description from the XML file and generate the components accordingly.
- **readTemplates()** Reads the templates values, that are described in the XML files and set it to the components, if the component is template-based.
- **connectComponents()** connects the components after identifying the relationships between the NICs (Network interface card) of the maschines and the ports of the switches.
- **readXML()** triggers the readTemplates(), readComponents() and connectComponents() and sets the global settings for the model. This method is the main method for reading a XML file that contains a model description.
- **writeXML()** is responsible for generating the XML file from the objects in the model.

5.2 View

The graphical user interface for the model is based on swing and awt. The main class that is responsible for the graphical user interface generation is called GUI.java that is a sub-class of the JFrame class of the swing library. The parts of the user interface are explained explicitly in the [Bra]. The reading of the above document is recommended in order to understand the user interface concepts, although the underlying model architecture and rendering methods were completely rewritten and extended during the internship.

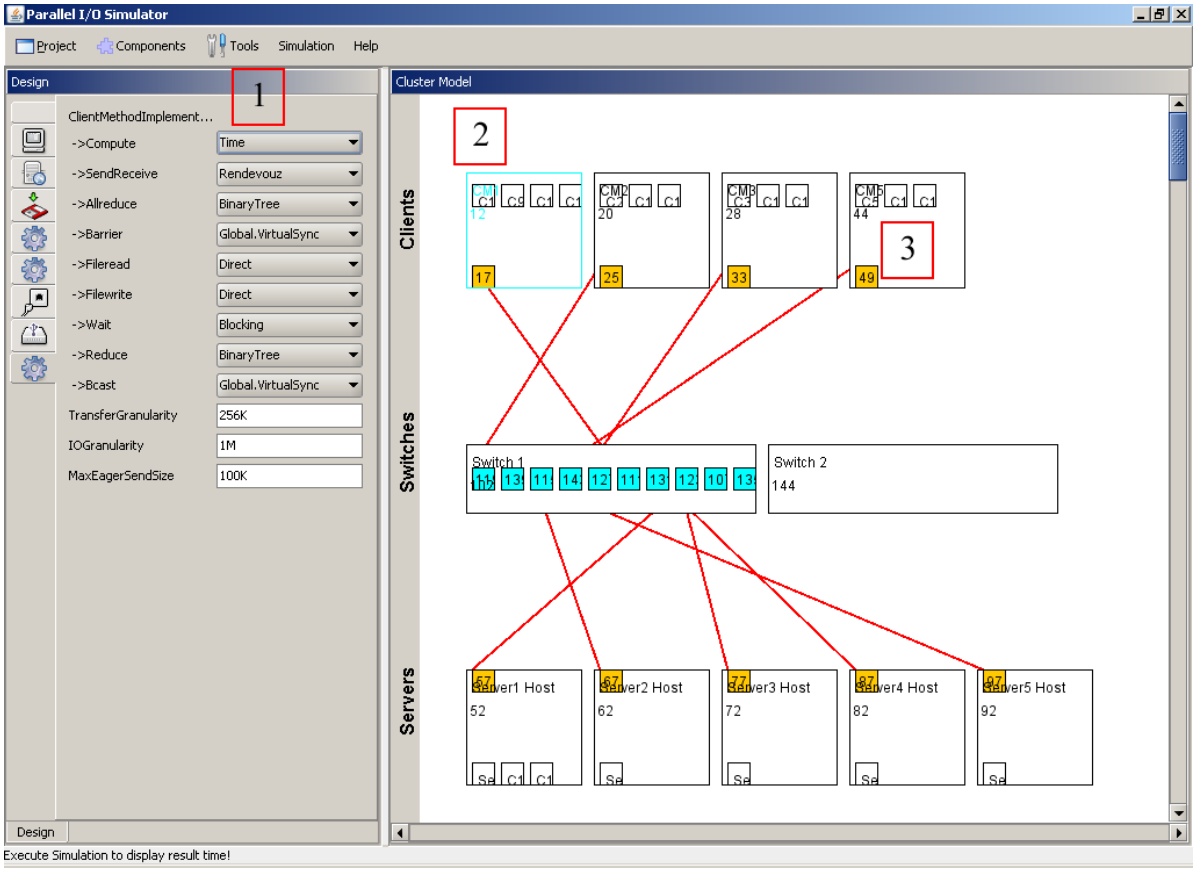

Figure 5.3: Graphical user interface of the Parallel I/O Simulator

Figure 5.3 shows the graphical overview of the parallel I/O simulator. Label [1] shows the design tab where the attributes of the components are set. Label [2] shows the JDrawingArea that is also a sub-class of the JPanel which is the container for all the graphical objects. Label [3] shows a maschine component, which in return holds a set of sub components such as clients and servers. Each Component inside the JDrawingArea implements the mouseListner() [A05] which acts as the controller in the MVC design architecture.

All the components in the cluster that have to be visualized in the user interface are coupled with a graphical object that is drawn in the Drawing Area of the GUI. GUIComponents object holds an instance of the model object and the GUI object. It acts as the interface between the

model and the view and generates corresponding gui objects from all the model objects. All the model objects are mapped using the following syntax for easy identification.

Table 5.1: Model Component mapping to GUI component

Component name	**GUI Component name**
Component	Component2D
Client.java	Client2D.java
Switch.java	Switch2D.java

All the GUI components are derived from the Component2D class which is a sub class of the JPanel. [1]

5.3 Functions

This section describes some of the extensions that were made to extend the functionalities of the user interface.

- **Annotation Concept**

  The annotation concept is introduced in the model to generate attributes for the dynamical left panel. An interface called AttributeGetters is defined to annotate the getter methods of the components. The following procedure shows how the annotation concept is used to generate dynamic attributes in the attributes panel depicted in figure 5.3, label [1].

  Assume that the attribute memorySize has to be visualized in the attribute panel. The user has only to annotate the method using the **AttributeGetters** interface. The **generateMenu()** method in the TabPanel class inside the GUI.java dynamically reads the annotation and sets the method as visible in the user interface.

```
1 @AttributeGetters
2         public long getMemorySize() {
3                 return memorySize;
4         }
```

- **Java reflection**
  Java reflection is an advanced java feature used for generic construction of classes during

[1] JPanel is a generic light weight container, used to define a rectangular space. See [A04] for how to use the panels

the runtime. For detailed information on the reflection application programming interface (API) see the [A08][DE]. Reflection interface is used to generate the class objects from the XML data file as shown in the following example. Please refer to the comments for further explanations.

```
<ComponentList>
        <MaschineList>
        <Maschine name="Server2_Host">
                <MemorySize>1073741824</MemorySize>
                <CacheSize>0</CacheSize>
                <InstructionPerSecond>1000</InstructionPerSecond>
                <InternalDataTransferSpeed>0</InternalDataTransferSpeed>
                <CPUs>1</CPUs>
                <NIC name="Server2_NIC1">
                        <Connection template="PVS-Connection" to="Switch_1">
                        <Bandwidth>122683392</Bandwidth>
                        <Latency>0.00002s</Latency>
                        </Connection>
                </NIC>
                <Server name="Server_2" template="PVS-Server">
                        <IOSubsystem name="Server_2_I/O-1" template="Std-Disk">
                        <AvgAccessTime>0.005s</AvgAccessTime>
                        <MaxThroughput>52428800</MaxThroughput>
                        </IOSubsystem>
                </Server>
        </Maschine>
        </MaschineList>
<SwitchList>
 ...
</SwitchList>
</ComponentList>
```

The generation of the class objects using reflection in the Model.java.

```
//Read the XML Components and generate the Component classes
private void readComponents(Element templateRoot) throws Exception {
    // component types e.g. Maschine, Client and Server
                for (Class<BasicComponent<?>> c : componentTypes) {
                  // get the name of the class
                        String className = c.getSimpleName();
                        // Read the Attributes  of the component type e.g.<
                            SwitchList>.
                        Element element = XMLutil.getFirstElementByTag(
                            templateRoot, className + "List");
                        // get all the components from the type
                        ArrayList<Element> list = XMLutil.getElementsByTag(element
                            ,     className);
                        for (Element e : list) {
                          // Generate the constructor
                                Constructor<BasicComponent<?>> ct = c.
                                    getConstructor(Model.class);
                                // Generate the component instance.
                                BasicComponent<?> component = ct.newInstance(this)
                                    ;
                                // Read the attributes for the component
                                component.readXML(e);
                        }
                }
        }
```

5.4 JPanel restrictions

A significant difficulty in the user interface is generating light weight lines. In the current model multiple switches are visualized in contrast to the original GUI where only vertical lines were allowed. In this case, a network cable (or connection) in the form of a line has to be drawn from two arbitrary positions in the JDrawingArea (see Figure 5.3). As the JDrawingArea is a sub-class of the JPanel, the components that are placed in the JDrawingArea has to be sub-classes of the JPanel to generate a uniform design model for all the visual components. Therefore, a new Component that is extended from the JPanel has to be defined for connection visualization. Figure 5.4 shows the design concept of the line inside a JPanel . Assume that a connection object has to be generate from the point $(x1, y1)$ to the point $(x2, y2)$. Then a JPanel object is generated to include a Line2D object from the package **java.awt.geom.Line2D** that extends from $(x1, y1)$ to $(x2, y2)$. Figure 5.4 shows the possible drawings of the lines in JPanel rectangles.

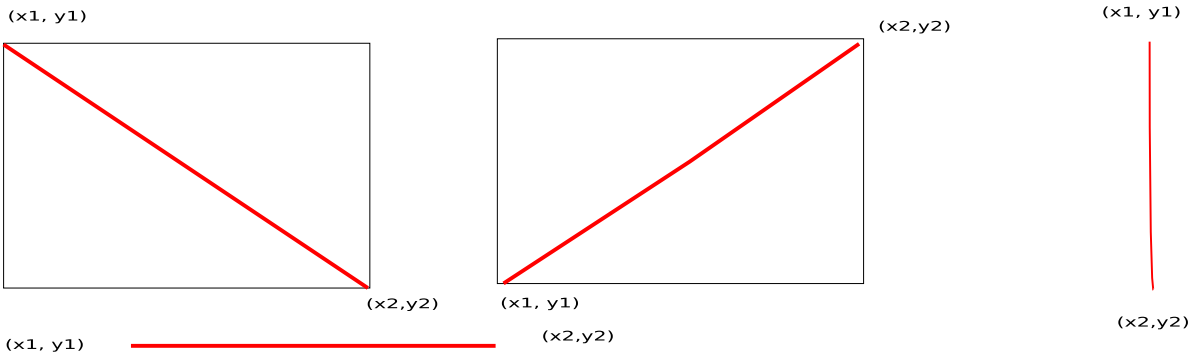

Figure 5.4: Line drawing inside a JPanel

As the figure 5.3 shows, there may be overlapped regions from several JPanels in the JDrawingArea. Overlapping makes it difficult to make the lines clickable because the JPanels overlap. A mechanism was developed to prevent the user from clicking in overlapped areas were generated using the distance to a line inside the JPanel. The following method in the Connection2D.java shows the implementation of the mouse change that is used to prevent the user from clicking the areas outside the line component.

```
1  addMouseMotionListener(new MouseMotionAdapter() {
2        @Override
3        public void mouseMoved(MouseEvent e) {
4          //if the distance from the line is smaller than 5 pixel, change the mouse to
               the cross cursor element from the default mouse pointer.
5          if (line.ptSegDist(e.getPoint()) < 5.0)
6            setCursor(myCrossCursor);
7          else
8            setCursor(myDefaultCursor);
9        }
10     });
11   }
```

The method for manipulating the mouse is not a perfect solution. It assumes that the JPanels are rendered in the correct order, if the clicking has to be correct. But in the current user interface, the component rendering is executed in an nondeterministic way and the user has no control over the rendering. Thus, the mouse hiding to detect the lines can be inaccurate in the long-run.

6 SWT vs. AWT

6.1 SWT Introdcution

SWT (Software Widget Toolkit) is an alternative user interface developing toolkit to the AWT/Swing Libraries. In this chapter, the basic functionalities of the SWT explained and the pros and cons of the SWT library is discussed.

SWT structure is based on three building blocks called shell, display and the widgets. SWT is a heavy weight GUI toolkit relying on the native operating systems drawing functionalities.

- Shell is the graphical window rendered by the window manager of the underlying operating system. Instance that do not have a parent object are generated in the shell.
- Display is responsible for event loop handling and managing the communication between the UI thread and the other threads.
- Widget is a basic interactive element in a graphical user interface. Example widgets are buttons, lists, text boxes and menus.

One of the advantages of the SWT is that it generally uses less memory thant the light weight toolkit Swing. The less memory usage is due to the fact that heavy weight toolkits pass most of the memory management to the native operating system rather than using its internal rendering techniques.

One of the other characteristics of a heavy-weight toolkit like swing is that it's tight coupling with native operating system. That makes each swt application appear similar to the operating system's native look. But Swing as a light weight toolikt has the advantage of looking similar in any operating system. Therefore the use case of PIOsim plays a major role in the selection of the GUI toolkit. Another setback of the SWT is that it simply wraps the gui functionalities that the operating system provides. Therefore customizing the widgets is complicated and sometimes impossible using SWT where Swing has a clear advantage. In swing there are lots of api elements that are provided with each gui object for customization purposes. The customization options make the debugging of the swing applications difficult, because of the large number of attributes.

In the context of performance, SWT is more responsive and uses less system resource than swing. But according to available performance benchmarks, there are differences that depend on the operating system and the java version the application is built on. See [A09] for the performance benchmarks of swt against swing.

6.2 The line drawing concept of SWT vs. AWT

To summarize the the AWT/Swing Concept, in a rectangular Jpanel is extend from the Swing component making it an interactable object in a JFrame component. As JPanel is a sub-class of the Container, it can hold a set of other components, which is a necessity in the piosim project. On the other hand, the Line component implemnts the Shape which is a class in the awt package. This makes the programming of a unique controller mechansim between the model and the graphical user interface difficult. When a new JPanel is defined for the line component and the line is painted inside it , this naturally puts a layer on the other components and the overlapping areas are unaccessible from the mouse pointer. Therefore the concept of sub-classing the JPanel for line drawing is only efficiently usable for non-overlapping lines.

SWT has a more generalized concept of drawing graphics than in AWT/Swing. , In SWT, there is a class called org.eclipse.swt.graphics.GC that includes the procedure for drawing lines and shapes. The following example shows the drawing of a line in a SWT shell. Some of the content contains code parts of [A14].

```
1  import org.eclipse.swt.*;
2  import org.eclipse.swt.events.*;
3  import org.eclipse.swt.graphics.Rectangle;
4  import org.eclipse.swt.widgets.*;
5  import org.eclipse.swt.layout.*;
6
7  public class DrawLine {
8    public DrawLine() {
9      Display display = new Display();
10     final Shell shell = new Shell(display);
11     shell.addPaintListener(new PaintListener() {
12       public void paintControl(PaintEvent e) {
13         Rectangle clientArea = shell.getClientArea();
14         e.gc.drawLine(0, 0, clientArea.width, clientArea.height);
15       }
16     });
17     shell.open();
18     while (!shell.isDisposed()) {
19       if (!display.readAndDispatch()) {
20         display.sleep();
21       }
22     }
23     display.dispose();
24   }
25   public static void main(String[] args) {
26     DrawLine d= new DrawLine();
27   }
28 }
```

7 Examples of better design

Users can download a set of sample applications from the Eclipse web site, that include the behavior of the widgets. A complete documentation of the examples (including the installation)is found under :

`http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.platform.doc.isv/samples/org.eclipse.swt.examples/doc-html/swt_manual_setup.html`

From the available examples, the paint example that is contained in the **org.eclipse.swt.examples.paint** can has most of the features that are needed for developing the PIOsim gui.

8 Conclusion

PIOsim GUI, which is intended for simulating clusters has to undergo several steps of change to become a reliable user friendly user interface. During the internship period, lots of efforts were made to develop the current swing gui in a more efficient graphical user interface. Although the basic integration and some of the necessary tasks were executed, there were lots of restrictions in the efficient visualizing.

Although the basic functionalities were integrated to the model, there are also some functional issues that has to be dealt with, that were explained in the current model chapter. The restrictions of the Swing API and the complexity of the Swing development dominated during the Internship period. Therefore the author suggests also looking to the SWT library before doing further development in to the GUI.

Bibliography

[A04] How to use panels. http://java.sun.com/docs/books/tutorial/uiswing/components/panel.html.

[A05] Mouselistener. http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Component.html#addMouseListener(java.awt.event.MouseListener).

[A07] The standard widget toolkit (swt). http://www.eclipse.org/swt/.

[A08] Reflection - die java reflection api. http://wiklet.javacore.de/index.php?oldid=448.

[A09] Swt vs. swing performance comparison. http://cosylib.cosylab.com/pub/CSS/DOC-SWT_Vs._Swing_Performance_Comparison.pdf.

[A10] Netbeans. http://www.netbeans.org/.

[A11] eclipse. http://www.eclipse.org/.

[A12] fireworks. http://www.adobe.com/de/products/fireworks/.

[A14] Swt graphics.

[Bra] Matthias Braun. Entwicklung einer grafischen oberflaechen fuer piosim.

[Ins] MageLang Institute. Java short course. http://java.sun.com/developer/onlineTraining/GUI/Swing2/shortcourse.html#JFCMVC.

[Mic] Sun Microsystems. Java BluePrints Model-View-Controller. http://java.sun.com/blueprints/patterns/MVC-detailed.html.