Entwicklung einer grafischen Oberfläche für PIOsim

Matthias Braun

matthiasbraun@maze2k.de

Betreuung an der Universität Heidelberg: Thomas Ludwig, Julian Kunkel, Stephan Krempel

Abstract

Diese Ausarbeitung behandelt die Entwicklung einer grafischen Oberfläche (GUI) eines Simulators für parallele Ein-/Ausgabe, welcher zeitgleich von zwei anderen Studenten entwickelt wurde. Es wird hierbei hauptsächlich auf die Implementierung in Java und die Struktur der Benutzeroberfläche eingegangen. Ebenso *werden die Möglichkeiten zur Erweiterung und Weiterentwicklung der GUI diskutiert. Die Arbeit wird durch Beispiele zur Bedienung der Oberfläche sowie anschauliche Abbildungen abgerundet.*

1. Einleitung

Die Entwicklung des Simulators für parallele Ein-/Ausgabe sowie die dazugehörige grafische Oberfläche, deren Implementierung das Thema dieser Arbeit sein wird, waren Bestandteil eines Softwarepraktikums für Fortgeschrittene am Lehrstuhl für parallele und verteilte Systeme an der Universität Heidelberg.

Die Zielsetzung dieses Praktikums war es, ein Cluster und seine Komponenten zu modellieren und anschließend den Ablauf von Programmen, welche parallele Ein-/Ausgabe als Inhalt haben, auf diesem modellierten Cluster zu simulieren. Das gewünschte Ergebnis sollten Zahlenwerte sein, welche die benötigte Ausführungszeit der (parallelen) Programme so real wie möglich widergeben sollten.

Diese Ausarbeitung beginnt mit der Beschreibung der Komponenten eines Clusters, die mit dem Simulator modelliert werden können. Danach folgt der Hauptteil der Arbeit, die Implementierung der grafischen Oberfläche und deren Beschreibung. Nach der Beschreibung folgt ein Beispiel für die Erstellung eines Cluster-Modells. Abschließend finden sich zwei Kapitel zur Erweiterung der bestehenden modellierten (Cluster-)Komponenten sowie zur Weiterentwicklung der Oberfläche.

2. Clusterkomponenten

Ein (Computer-) Cluster ist eine Anzahl von miteinander vernetzten Rechnern mit der Zielsetzung, beispielsweise die Rechenleistung oder die Verfügbarkeit gegenüber einem einzelnen Rechner deutlich zu erhöhen. In unserem Simulator besteht ein solcher Cluster aus fünf verschiedenen Komponenten:

- Clients
- Server
- Netzwerkverbindungen
- Switches
- Festplatten

Im Folgenden werden die einzelnen Komponenten kurz beschrieben.

Ein Client ist ein Rechner, auf dem ein (paralleles) Programm ausgeführt wird. Er ist üblicherweise mit einem Switch verbunden.

Ein Server ist ein Rechner, der in unserem Modell dazu dient, Daten auf eine oder mehrere Festplatten zu speichern. Er ist üblicherweise ebenfalls mit einem Switch verbunden, sowie zusätzlich mit einer internen oder externen Festplatte.

Ein Switch verbindet mehrere Rechner miteinander, in unserem Modell die Clients und die Server.

Die Netzwerkverbindung stellt die Verbindung zwischen zwei Komponenten dar, in unserem Fall zwischen Server und Switch bzw. Client und Switch.

Jede dieser Komponenten besitzt spezifische Attribute, die deren Leistungsfähigkeit genauer eingrenzen. So gibt es zum Beispiel in allen Komponenten eine Latenzzeit. Je nachdem, um welche Komponente es sich handelt gibt es

beispielsweise noch Attribute wie das ausgeführte Programm, den Rang, die Bandbreite, die Anzahl Instruktionen pro Sekunde, die Größe des Cache, die Größe des Arbeitsspeichers, usw.

Abbildung 1 zeigt eine mögliche Konstellation aus diesen verschiedenen Komponenten.

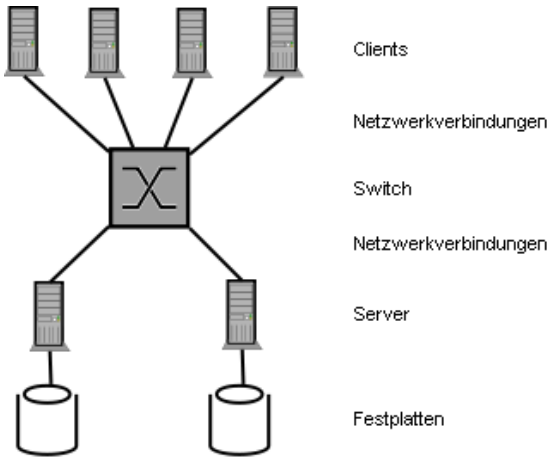

Abbildung 1 – Beispielkonstellation eines Clusters

Prinzipiell bräuchte man nicht notwendigerweise die Unterscheidung zwischen Client und Server, jedoch ist die Modellierung der Ein-/Ausgabe mit dieser Sicht auf ein Cluster einfacher zu gestalten.

3. Implementierung und Struktur

Die Oberfläche des Simulators sollte genauso wie der Simulator selbst in Java implementiert werden. Bevor mit der Implementierung begonnen wurde, mussten zuerst einige Entscheidungen getroffen werden. Dies war einmal die Wahl der verwendeten Java Bibliothek zur Oberflächenprogrammierung. Hier entschied ich mich aus Gründen der Optik und meiner bisherigen Erfahrungen zu der Swing Bibliothek und der etwas kompakteren JGoodies Bibliothek.

Danach musste eine Screen-Struktur entwickelt und durch die Betreuer abgesegnet werden. Diese wurde auch bis zum Ende beibehalten und ist später in diesem Kapitel noch in verschiedenen Abbildungen zu sehen.

Schließlich musste noch darüber entschieden werden, welche Attribute jede Komponente hat. Diese Attribute wurden in Kapitel 2 schon in ihren deutschen Bezeichnungen genannt und werden hier nur noch kurz den Komponenten zugeordnet:

- Client: Name, MemorySize, CacheSize, InstructionsPerSecond, BandWidth, Latency, Program, Rank
- Server: Name, MemorySize, CacheSize, InstructionsPerSecond, BandWidth, Latency, Disk
- Netzwerkverbindung: Name, BandWidth, Latency
- Switch: Name, TotalBandwidth, Bandwidth, Latency
- Festplatte: Name, AvgAccessTime, MaxThroughput

Nachdem diese Grundlagen festgelegt wurden, konnte der erste Prototyp der GUI implementiert werden. Dieser war optisch schon beinahe identisch mit der aktuellen Version der Oberfläche, jedoch gab es einige programmiertechnische Schwierigkeiten, die ein Re-Engineering nötig machten.

Im Folgenden werden nun zuerst die einzelnen Klassen der GUI vorgestellt, danach folgt ein Beispiel, welches die komplette Modellierung eines kleinen Clusters mit der Oberfläche Schritt für Schritt zeigt.

Zuerst wurden die Klassen implementiert, welche die einzelnen Komponenten logisch repräsentieren. Diese befinden sich im Package de.hd.pvs.piosim.components:

- *Client*
- *Server*
- *Switch*
- *NetworkConnection*
- *HardDisk*

Diese Klassen erben alle von der Klasse *Component* und besitzen alle einen Array von *Attribute*-Instanzen, welche die schon genannten Attribute repräsentieren. Die Klasse *Attribute* beinhaltet außerdem einige Hilfsmethoden, um mit den *Attribute*-Arrays besser und einfacher umgehen zu können.

Die Komponentenklassen sind momentan eigentlich nur Platzhalter für die „richtigen“ Komponentenklassen aus den Simulatoren, da leider zum Zeitpunkt der Abgabe dieser Arbeit der Simulator bzw. die beiden parallel entwickelten Simulator-Prototypen nicht komplett fertiggestellt wurden. Hierzu aber nachher mehr im Kapitel Weiterentwicklung.

Die Klassen für die grafische Repräsentierung der Komponenten sowie die Klasse für die GUI selbst befinden sich im Package de.hd.pvs.piosim.gui. Zuerst die Klassen, welche die einzelnen Komponenten grafisch darstellen:

- *ClientIntf*
- *ServerIntf*
- *SwitchIntf*
- *ConnectionIntf*

Die Festplatten-Komponente wird nicht grafisch repräsentiert. Sie gehört logisch zum Server und wird nur über dessen Attribute dargestellt.

All diese Klassen erben von der *JPanel*-Klasse und überschreiben deren *paint()*-Methode. Darin wird ein Bild der jeweiligen Komponente gezeichnet und gegebenenfalls mit einem Namen versehen.

Ebenfalls in diesem Package enthalten ist die Klasse *JDrawingArea*, welche die Grundfläche für die Darstellung des Cluster-Modells bietet. Sie ist unterteilt in drei Bereiche: Oben der Client-Bereich, in der Mitte der Switch-Bereich und unten der Server-Bereich. Getrennt sind diese Bereiche durch zwei gestrichelte Linien.

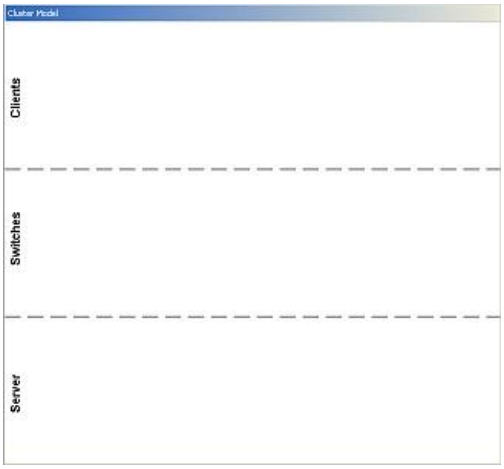

Abbildung 2 - JDrawingArea

Bevor nun die Hauptklasse *GUI* etwas detaillierter vorgestellt wird, sollen noch kurz die restlichen Klassen in diesem Package erwähnt werden:

- *CompAttr*
- *Diagram*

Die Klasse *CompAttr* ist eine Hilfsklasse, die eine Verbindung zwischen den Eingabefeldern auf der Oberfläche und den Attributen herstellt. Die Klasse *Diagram* wird später noch einmal kurz vorgestellt.

Die letzte und wichtigste Klasse in diesem Package ist die Klasse *GUI*. In ihr ist die gesamte sichtbare Oberfläche implementiert. Die Darstellung der verschiedenen Bereiche ist durch unterschiedliche Methoden gekapselt, sodass man diese jederzeit leicht ändern kann.

Abbildung 3 zeigt die Unterteilung der Oberfläche in 4 Bereiche:

- Menuleiste (oben)
- Designmenu (Mitte links)
- Zeichenfläche (Mitte rechts)
- Statusleiste (unten)

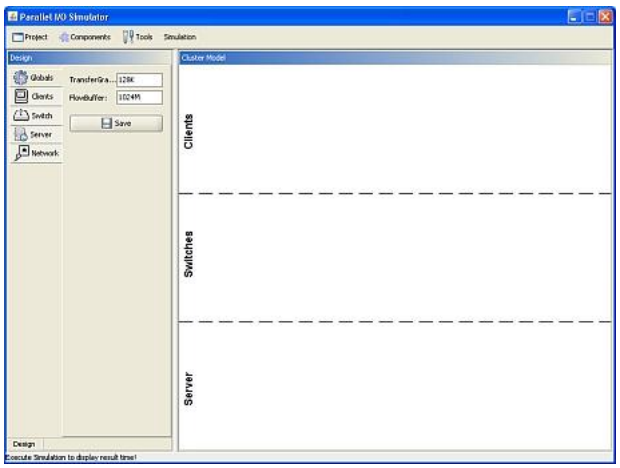

Abbildung 3 - Screenshot der Oberfläche

Die Methode *GUI.buildMenuBar()* erzeugt die Menuleiste im oberen Bereich der Anwendung. In diesem Menu befinden sich vier Menubereiche:

- Projektmenu (Project)
- Komponentenmenu (Components)
- Werkzeugmenu (Tools)
- Simulationsmenu (Simulation)

Im Projektmenu kann man ein neues Projekt erstellen, ein Projekt öffnen, das aktuelle Projekt speichern oder schließen, (parallele) Programme sowie Templates importieren und das Programm beenden.

Über das Komponentenmenu öffnet man im Designmenu die Eingabemaske für das Erstellen der jeweiligen Komponente. Man kann hier alle Arten von Komponenten anlegen: Clients, Switches, Server und Netzwerkverbindungen.

Das Werkzeugmenu beinhaltet nur einen Eintrag: den Template und Programm-Manager (Template & Program Manager), über den man sich die importierten Templates und Programme anzeigen lassen kann. Man hat hier außerdem die Möglichkeit, die vorhandenen Templates zu ändern. Abbildung 4 zeigt den Template und Programm-Manager.

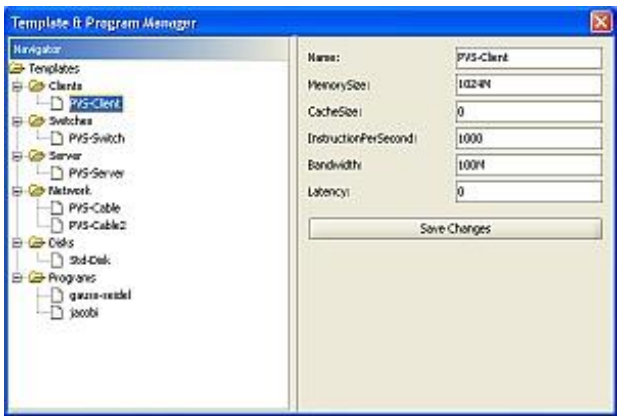

Abbildung 4 - Template und Programm-Manager

Die Funktionalität des Simulationsmenus ist derzeit nicht bzw. nur prototypisch vorhanden. Da die Implementierung des eigentlichen Simulators zum Zeitpunkt dieser Arbeit nicht komplett fertiggestellt war, beinhaltet der Menupunkt zum Starten der Simulation (Simulate) noch keine Funktion. Die zweite Funktion in diesem Menu ist die Anzeige der Diagramme (Show Diagrams). Nach dem Durchlauf der Simulation ermöglicht es dieser Menupunkt, an die bereits erstellten Komponenten ein Diagramm in Miniatur-Ansicht anzuhängen, welches die Auslastung der Komponente im Verhältnis zur gesamten simulierten Zeit darstellt. Ein Klick auf dieses Diagramm auf der Zeichenfläche öffnet eine vergrößerte Version des Diagramms. Abbildung 5 zeigt ein Beispiel-Modell eines Clusters auf der Zeichenfläche mit angehängten Diagrammen, Abbildung 6 die vergrößerte Darstellung eines Diagrammes. Die Werte auf diesen Diagrammen sind im momentanen Stand der Implementierung noch durch Zufallszahlen gegeben, da noch keine exemplarischen Ergebnisse der Simulation vorlagen.

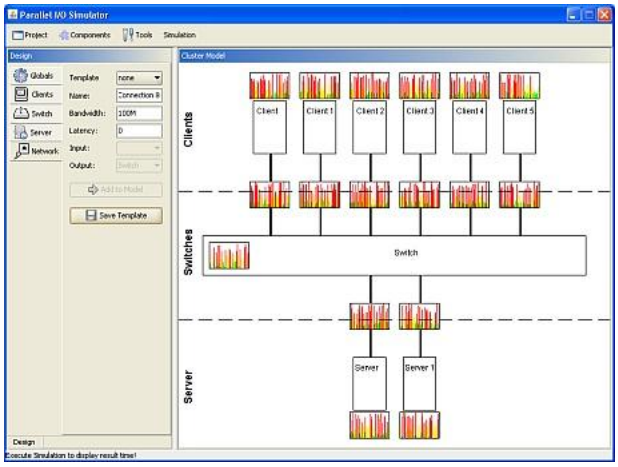

Abbildung 5 - Modell mit Diagrammen

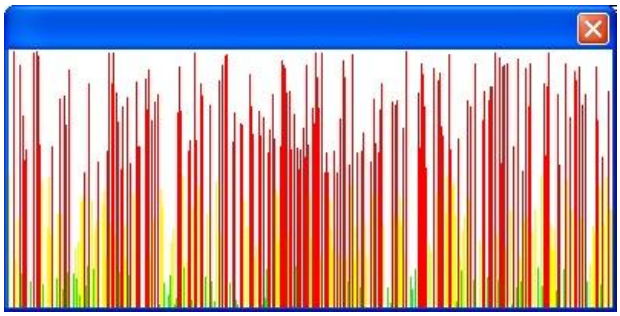

Abbildung 6 - Diagramm (vergrößerte Darstellung)

Der letzte Menupunkt im Simulationsmenu ist die Anzeige des Diagram Viewers. Dieser zeigt die Diagramme aller Komponenten übereinander angeordnet an. Somit ist ein Vergleich der Auslastungen möglich und man kann den Ablauf der Simulation noch genauer analysieren. Abbildung 7 zeigt einen Screenshot des Diagram Viewers mit den zufällig erzeugten Diagrammwerten des Cluster-Modells aus Abbildung 5.

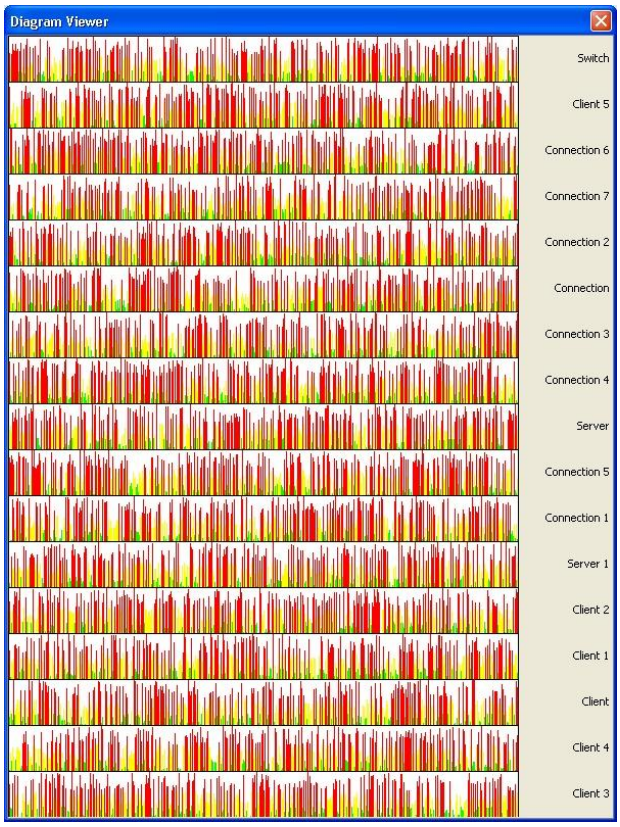

Abbildung 7 - Diagram Viewer

Der zweite große Anzeigebereich in der Oberfläche des Simulators ist das Designmenu. Es besteht aus fünf Tabs, über die man die Menus zum Setzen der globalen Einstellungen sowie dem Erstellen der vier Komponententypen erreicht. Implementiert ist die Anzeige des Menus in der Methode *GUI.buildMenu()*. Hierzu gehören auch die Methoden *GUI.buildGlobalAttributesPanel()* zum Erstellen der Eingabe der globalen Einstellungen und *GUI.buildAttributesPanel()*. Letztere Methode erstellt die Eingabeflächen, über welche man die einzelnen Komponenten neu erstellt bzw. vorhandene ändert und aktualisiert.

Abbildung 8 zeigt die Eingabemaske für einen neuen Client. Man hat hierbei die Möglichkeit ein Template auszuwählen, das die Eingabefelder mit entsprechenden Vorgabewerten füllt. Anschließend kann man die einzelnen Attribute-Felder ausfüllen und die Komponente abspeichern. Die Eingabefelder dieser Attribute (außer dem Template) werden aus den schon erwähnten *Attribute*-Arrays dynamisch erzeugt. Abschließend kann man die Komponente dann mittels Klick auf die Schaltfläche dem Modell hinzufügen, bzw. wenn man eine Komponente gerade ändert kann

man die Komponente durch Klick auf die entsprechende Schaltfläche aktualisieren.

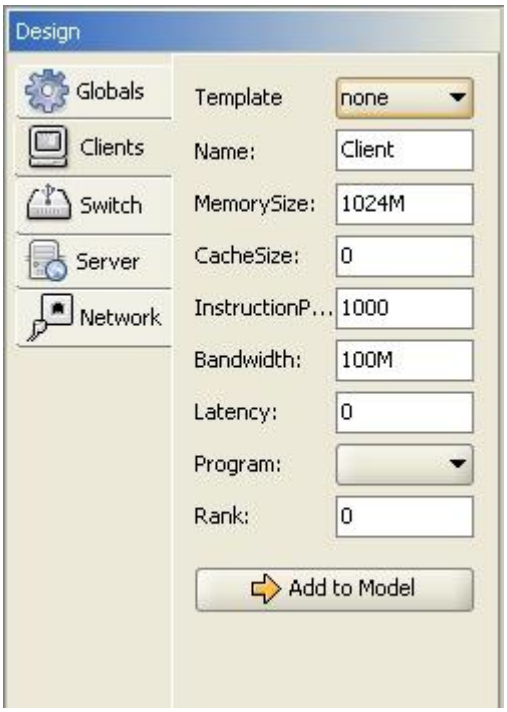

Abbildung 8 - Designmenu: Erstellung eines Clients

Die Komponenten, die man im Designmenu erstellen kann, werden auf der Zeichenfläche dargestellt. Diese wird in der Methode *GUI.buildMainRightPanel()* erstellt. In Abbildung 9 sieht man die Beispielkonfiguration von vorhin ohne die eingeblendeten Diagramme. Man erkennt dort deutlich die Darstellungen der einzelnen Clients, Server, Netzwerkverbindungen und des Switches.

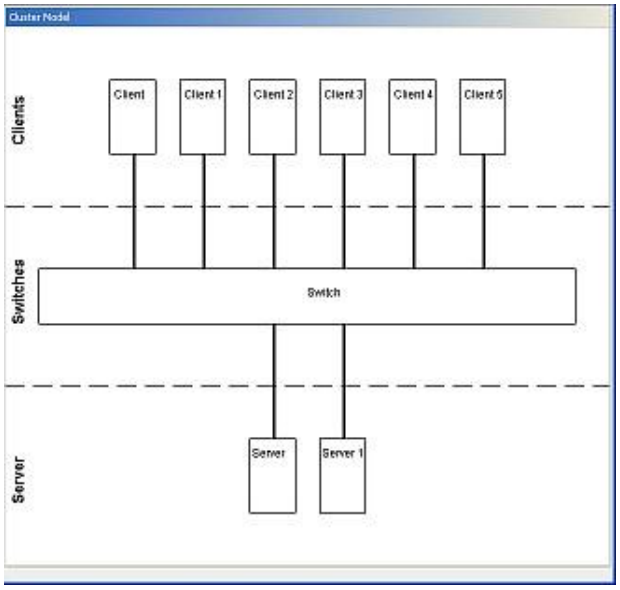

Abbildung 9 - Darstellung der Komponenten

Der letzte Bereich auf der Oberfläche ist die Statusleiste. Hier können Informationstexte angezeigt werden, z.B. die Simulationszeit und/oder die simulierte Zeit. Sie wird in der Methode *GUI.buildStatusBar()* realisiert und ist über die Variable *statusbar* überall in der Applikation ansprechbar.

4. Modellierung eines Cluster-Modells

In diesem Kapitel wird in kurzen Worten erläutert, wie das schon mehrmals genannte Cluster-Modell in den Abbildungen 5 und 9 erstellt werden kann. In welcher Reihenfolge man die einzelnen Komponenten erstellt, ist für das Modell und die Simulation nicht wichtig. Die einzige (logische) Beschränkung beim Modellieren liegt darin, dass zuerst ein Switch und ein Client bzw. Server vorhanden sein müssen, damit eine Netzwerkverbindung erstellt werden kann.

Schritt 1: Konfiguration des Simulators.

Nach dem Starten der Simulator-Oberfläche ist der in Abbildung 8 sichtbare Tab *Globals* aktiviert. Hier kann man die globalen Attribute wie z.B. die *Transfer Granularity* und den *Flow Buffer* festlegen.

Zusätzlich kann man, wenn nötig, über den *Template & Program Manager* die Default-Templates editieren. Man klickt hierzu in der Menuleiste auf *Tools* und dann auf *Template & Program Manager*. Im nun geöffneten Fenster kann man die einzelnen bereits vorhandenen Templates anklicken und die Attributwerte darin ändern.

Um eine Simulation starten zu können, müssen auch noch Programme importiert werden. Diese müssen im XML-Format bereits vorliegen und können über die Menuleiste importiert werden, indem man auf *Project* und dann auf *Import Program* klickt. In dem Dialogfenster kann man wie aus anderen Anwendungen gewohnt die gewünschte Datei auswählen und importieren. Hat man verschiedene Programme, muss man diesen Vorgang mehrmals wiederholen.

Schritt 2: Erstellen der Clients

Zum Erstellen der Clients klickt man auf den in Abbildung 8 gezeigten Tab *Clients*. Hier werden für jeden Client eventuell ein Template und die Attributwerte eingegeben bzw. verändert. Zu diesen Attributwerten gehört insbesondere das Attribut *Program*, über welches das auszuführende (parallele) Programm gewählt wird sowie den Rang (Attribut *Rank*) in diesem Programm. Abschließend klickt man auf die Schaltfläche *Add to Model*, um die Komponente auf der Zeichenfläche anzulegen.

Schritt 3: Erstellen des Switches

Um den Switch zu erstellen, klickt man auf den Tab *Switch*, der sich direkt unter dem Tab *Clients* befindet. Hier geht man ebenso vor wie beim Erstellen der Clients: Zuerst wird wenn gewünscht ein Template gewählt, dann die Attributwerte wenn nötig geändert und abschließend auf *Add to Model* geklickt.

Schritt 4: Erstellen der Server

Die Erstellung der Server geschieht analog zum Erstellen der Clients in Schritt 2 bis auf den Unterschied, dass hier kein *Program* und *Rank* ausgewählt werden, dafür eine *Disk*. Zum Modell hinzugefügt wird die Server-Komponente ebenfalls über einen Klick auf *Add to Model*.

Schritt 5: Erstellen der Netzwerkverbindungen

Sind nun alle anderen Komponenten erstellt, kann man sie durch Netzwerkverbindungen miteinander verbinden. Dazu klickt man auf den Tab Network und wählt hier auch wieder das Template, füllt die Attributwerte aus und wählt dann zusätzlich noch die *Input*- und die *Output*-Komponente. Mit *Add to Model* wird dann auch die Netzwerkverbindung dem Modell hinzugefügt.

Nun sollte das Modell auf der Zeichenfläche genauso aussehen wie auf Abbildung 9 gezeigt. Der Beispiel-Cluster ist nun fertig modelliert und kann an den Simulator übergeben werden. Angenommen, die Logik des Simulators wäre implementiert und mit der Oberfläche verbunden, könnte man nun über die Menuleiste durch Klicken auf *Simulation* und anschließend auf *Simulate* die Simulation starten. Nach der Simulation kann man im gleichen Menu über die Einträge *Show Diagrams* bzw. *Diagram Viewer* die entstandenen Ergebnisse noch visualisieren. Diese beiden Funktionen sind wie bereits beschrieben durch zufällig generierte Ergebniswerte bereits implementiert und können angezeigt werden, wie in Abbildungen 5 und 7 zu sehen.

5. Erweiterung

Bei der Implementierung der Oberfläche wurde stets darauf geachtet, dass sie an vielen Stellen ohne großen Aufwand erweiterbar ist. In diesem Kapitel werden diese Erweiterungsmöglichkeiten nun vorgestellt.

Der wichtigste Erweiterungspunkt liegt in den Implementierungen der einzelnen Komponenten. Man kann hier sehr einfach neue Attribute oder sogar Attributtypen hinzufügen. Hierzu muss man den Quellcode der jeweiligen Komponente im Package *de.hd.pvs.piosim.components* öffnen. Hier findet man die Variable attributes, in der alle Attribute untereinander aufgelistet sind. Möchte man ein neues Attribut hinzufügen, dessen Typ bereits existiert, muss man nur zwei Dinge tun:

- Ein neues Element in den *Attribute*-Array hinzufügen (*TEXT* steht hier nur demonstratv für einen vorhandenen Typ):
```
new Attribute("NeuesAttribut", "DefaultWert", Attribute.TEXT)
```
- Im Quellcode der Klasse Attribute muss man in der Variable *IDMapper* ein neues Element am Ende einfügen:
```
"NeuesAttribut"    // ID
```

Soll ein Attribut mit einem neuen Typ hinzugefügt werden, muss man vor den oben genannten einige zusätzliche Änderungen vornehmen:

- In der Klasse Attribute muss eine neue statische Variable für den Typ angelegt werden:
```
public static final int NEUERTYP = 123;
```
Wobei statt 123 hier ein *int*-Wert gewählt werden sollte, der bisher noch für keine statische Variable vergeben wurde.
- In der statischen Methode *Attribute.getType(String name)* muss in der if-Schleife analog zu den bereits vorhandenen Einträgen für den neuen Typ ein neuer Eintrag implementiert werden:
```
} else if (name.equals("NeuesAttribut")) {
return NEUERTYP;
```
- Sollte es sich um ein komplexeres Attribut handeln, d.h. soll man in der Oberfläche etwas anderes als ein Texteingabefeld haben, beispielsweise eine *Combobox* oder etwas anderes, muss man in der Methode *GUI.buildAttributePanel()* in den switch-Blöcken noch die jeweiligen Implementierungen dafür hinzufügen.

Hat man die neuen Attributtypen implementiert, muss man selbstverständlich auch noch die Algorithmen im eigentlichen Simulator anpassen, was jedoch hier nicht weiter beschrieben werden kann und soll.

Möchte man die grafische Darstellung der einzelnen Komponenten ändern, kann man dies auch sehr einfach durchführen. Man muss dazu nur die Methode *paint(Graphics g)* der jeweiligen Oberflächenklasse im Package *de.hd.pvs.piosim.gui* (Endung: *Intf*) anpassen. Man könnte beispielsweise das Rechteck für einen Client in einen Kreis umwandeln. Hierzu müsste

man folgendes ändern:

```
g.drawRect(0, 0, WIDTH-1, HEIGHT-1);
```

wird einfach ersetzt durch:

```
g.drawOval(0, 0, WIDTH-1, HEIGHT-1);
```

Und schon erhält man statt der rechteckigen Darstellung eine ovale. Selbstverständlich kann man hier mit etwas Kreativität komplexere und optisch ansprechendere Darstellungen implementieren. Auch dies soll aber nicht Gegenstand dieser Arbeit sein, sondern bleibt denen überlassen, die sich in Zukunft mit der Darstellung der Komponenten beschäftigen werden.

Selbstverständlich kann die Oberfläche noch an vielen weiteren Stellen verändert und erweitert werden. All diese zu nennen und näher zu beschreiben würde jedoch über den Umfang dieser Arbeit hinausgehen.

6. Weiterentwicklung

In diesem Kapitel sollen im Gegensatz zum vorherigen nicht die Möglichkeiten zur Erweiterung und Änderung der Oberfläche erörtert, sondern vielmehr ein Ausblick auf zukünftige Entwicklungen der momentanen Implementierung gegeben werden.

Die wichtigste Weiterentwicklung, die im Moment absehbar ist, ist die Zusammenführung des eigentlichen Simulators mit der grafischen Oberfläche. Hierzu muss man nur die logischen Repräsentationen der Komponenten im Package *de.hd.pvs.piosim.components* durch die tatsächlichen Implementierungen ersetzen. In diese muss man dann noch die Vererbung von der Klasse *Component* einfügen, sowie die statische Variable *attributes* und die *getAttributes()* Methode. Man könnte die aktuelle Implementierung auch so lassen, wie sie ist, jedoch hätte man dann einmal die Implementierung der Komponenten-Logik für den Simulator, und dann noch die momentanen Implementierungen, die nur die Attribute beinhalten. Dies könnte man durch die oben genannte Methode etwas kompakter gestalten.

Zu dieser Zusammenführung gehört auch, die Möglichkeit im Menu, die Simulation zu starten, mit Leben zu füllen. Dazu muss man lediglich in der Methode *GUI.buildMenuBar()* beim entsprechenden *JMenuItem* den *ActionListener* implementieren und darin die Attribute der Komponenten in geeigneter Art und Weise an die Simulatorlogik übergeben.

Die nächste Stufe der Weiterentwicklung wäre es, die Anzeige der Diagramme bzw. des Diagram Viewers mit den richtigen, vom Simulator erzeugten Ergebnisdaten aufzurufen. Diese Werte müssen an einer Stelle im Programm nach der Simulation gespeichert werden und in der Methode *GUI.drawDiagrams()* an den Konstruktor der Diagramme übergeben werden. Auch muss man dann eine geeignete Speicherstruktur für die Diagramme auswählen, da diese momentan immer neu erzeugt werden und nicht persistent in der Applikation vorhanden sind.

Weitere Ergebnisdaten der Simulation können dann auch in der Statusleiste angezeigt werden. Dies sind zum Beispiel die Anzahl Jobs, Datenpakete, die Simulationszeit oder die simulierte Zeit. Die Statusleiste ist über die Variable *statusbar* ansprechbar.

Sind diese Weiterentwicklungen durchgeführt, ist die Simulationsoberfläche erst einmal komplett. Man kann dann das Cluster-Modell erstellen, es an den Simulator übergeben, die Programme simulieren und das simulierte Ergebnis abschließend über Diagramme und Textausgabe in der Statusleiste visualisieren.

Selbstverständlich gibt es jetzt noch unzählige Möglichkeiten, die Oberfläche weiterzuentwickeln. Denkbar ist zum Beispiel die Möglichkeit, die Zeichenfläche als Bilddatei, beispielsweise im JPG oder PNG-Format abspeichern oder ausdrucken zu können.

Möglich wäre auch eine Weiterentwicklung hinsichtlich des Cluster-Modells. Momentan ist hierbei eine sehr enge Struktur vorgegeben: Maximal acht Clients, ein Switch und maximal acht Server. Man könnte hier grafische Möglichkeiten entwickeln, um die Beschränkung auf acht Komponenten einer Art aufzuheben. Ebenso könnte man die Beschränkung auf einen Switch aufheben und komplexere Cluster mit mehreren Switches modellieren.

Denkbar ist auch die Einbindung von Visualisierungstools, mit denen man die vom Simulator erzeugten Ergebnisse extern visualisiert bzw. bereits vorhandene Bibliotheken verwendet, um die grafische Ergebnisausgabe der Oberfläche zu erweitern bzw. zu ersetzen.

7. Zusammenfassung

Die grafische Oberfläche für PIOsim ermöglicht es also, einfache Cluster-Modelle mit einem Switch und jeweils maximal acht Clients und Servern mit den dazugehörigen Netzwerkverbindungen und Komponenteneigenschaften zu modellieren. Dabei kann man vorgefertigte Templates verwenden oder die Komponenten nach eigenen Vorstellungen frei

konfigurieren. Die Oberfläche umfasst zusätzlich die Funktionalität, die simulierten Ergebnisse mittels Diagrammen und Textausgabe in einer Statusleiste zu visualisieren. Der momentane Stand der Dinge ist, dass die Simulations-Logik noch nicht mit der grafischen Oberfläche zusammengeführt wurde und daher die Simulation noch nicht aus der Oberfläche heraus aufgerufen werden kann. Jedoch ist es durch die Architektur der GUI leicht möglich, die Logik des Simulators zu integrieren und die Applikation damit zu vervollständigen.