Entwurf und Implementierung einer graphischen Oberfläche für den Ulix-Emulator

Bachelorarbeit
von
Sichol Thanormsuay

Vorgelegt am
Lehrstuhl für Praktische Informatik 1
Prof. Dr. Felix Freiling
Universität Mannheim

Betreuer: Prof. Dr. Felix Freiling

Dezember 2008

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegeben Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mannheim, den 15. Dezember 2008 **Sichol Thanormsuay**

Kurzfassung

Die Idee für den Ulix-Emulator kam von Prof. Dr. Felix Freiling, um dieses Program als Lehrmittel in der Vorlesung „Betriebssystem“ an der Universität Mannheim benutzen zu können. Der Ulix-Emulator sollte im Konzept „MVC“[1] als Architekturmuster entworfen werden, damit spätere Änderungen oder Erweiterungen erleichtert werden.

In dieser Bachelorarbeit wurde die graphische Oberfläche für den Ulix-Emulator implementiert, und ich benutzte NetBeans IDE [2] als Entwicklungsumgebung.

Inhaltsverzeichnis

1. Einleitung

1.1 Motivation und Kontext

Ulix ist ein Betriebssystem, das aus Compilerteil und Emulatorteil besteht. Die Idee zu diesem Betriebssystem wurde von Prof. Felix Freiling konzipiert, um dieses Betriebssystem als Lehrmittel in der Vorlesung „Betriebssysteme“ an der Universität Mannheim benutzen zu können.

1.2 Aufgabenstellung

GUI (Graphical User Interface) für den Emulator muss erstellt werden, indem es unterschiedliche Hierarchie mit Emulatorteil ist. GUI für den Emulator muss mit Java-Sprache implementiert werden, damit es über unterschiedliche Betriebssysteme ausgeführt werden kann.laufen kann.

1.3 Ergebnisse

GUI für den Emulator ist fertig erstellt, es wurde mit NetBeans IDE[2] als integrierte Entwicklungsumgebung implementiert. Wesentliche Klassen, die von GUI für den Emulator benutzt werden, sind javax.swing.JComponent, javax.swing.AbstractButton und java.lang.Thread

2. Was ist Ulix?

2.1 Einleitung

Ulix ist ein Betriebssystem, das von Prof. Felix Freiling entwickelt wurde, um es in der Vorlesung „Betriebssysteme“ an der Universität Mannheim zu benutzen.
In diesem Kapital konkretisiere ich, wie das Ulix-Betriebssystem und die Ulix-Hardware aussehen.

2.2 Ulix-Betriebssystem

Das Ulix-Betriebssystem besteht aus 2 Hauptkomponenten:

- Ulix Compiler
  Quellcode wird in C-Sprache geschrieben und wird zu Img-Datei kompiliert.
- Ulix Emulator
  Die Img-Datei wird durch den Ulix Emulator simuliert und durch Ulix-GUI dargestellt.

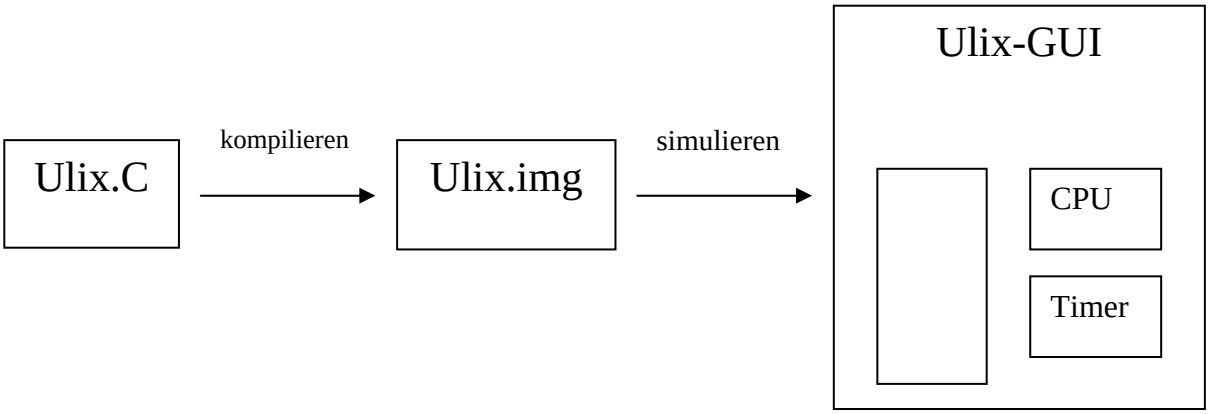

Abbildung1: Ulix-Betriebssystem

Ulix-Kernel wurde in C-Sprache geschrieben (siehe Abbildung 1), dann wird Ulix.c zu Ulix.img kompiliert, damit es über Ulix-Hardware ausgeführt werden kann.

Danach man kann diese Ulix.img durch den Ulix-Emulator simulieren, und es kann über Ulix-GUI dargestellt und kontrolliert werden.

Was wird auf Ulix-GUI dargestellt und kontrolliert?

CPU n : Register R0 bis R15, USP, SSP, PC, PSW,
M, Z, C1, C2, IRR, IER, IVT

Timer n : PTRU, PTSU, PTPCR, PTCR

Memory : Alle Werte, die im Hauptmemory gespeichert sind.

I/O-Controller: Alle wesentlichen Register von I/O-Controller

2.3 Ulix-Hardware

Es besteht aus 4 Hauptkomponenten CPU, TIMER, I/O-Controller und Mainmemory (siehe Abbildung 2).

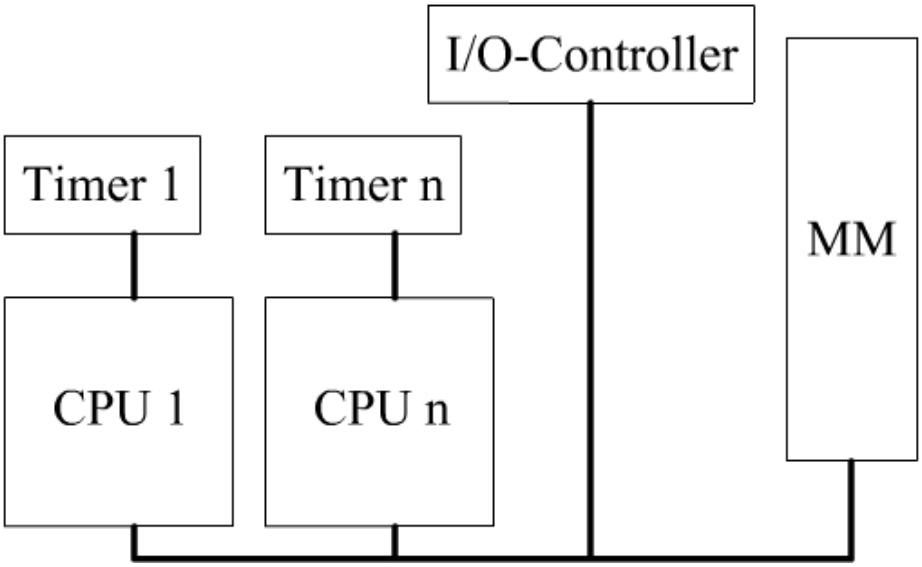

Abbildung 2: Ulix

Durch Ulix-GUI wird eine Komponente ausgewählt, die durch den Command-Befehl ausgeführt wird, aber gleichzeitig darf nicht mehr als eine Komponente ausgeführt werden.

2.4 Zusammenfassung

In diesem Kapital habe ich erläutert, was genau Ulix ist. Im nächsten Kapital werde ich speziell auf die Ulix-GUI eingehen.

3. Benutzung des Ulix GUI

3.1 Einleitung

Ich werde in diesem Kapital den Aufruf des Emulators, alle Menüs des Ulix-GUI und die Ablaufoptionen des Ulix-GUI verdeutlichen.

3.2 Aufruf des Emulators

Ulix-GUI kann durch „C:\java MainPage Args[0] Args[1], Args[2]“ aufgerufen werden, Args[] sind die Parameteroptionen, die man eingeben kann, bevor das Ulix-GUI gestartet wird.

- Args[0] = Configdatei (Standardeinstellung = ulix.conf)
- Args[1] = Imagedatei (Standardeinstellung = ulix.img)
- Args[2] = Diskdatei (Standardeinstellung = ulix.dsk)

Falls ulix.conf (als Standardeinstellung) nicht existiert, werden die Standardkomponenten als Standardeinstellung übernommen.

- CPU = 1
- Memory = 8192
- Disksize = 100
- Sektorsize = 512

Anschließend wird dieses Fenster angezeigt (siehe Abbildung 2).

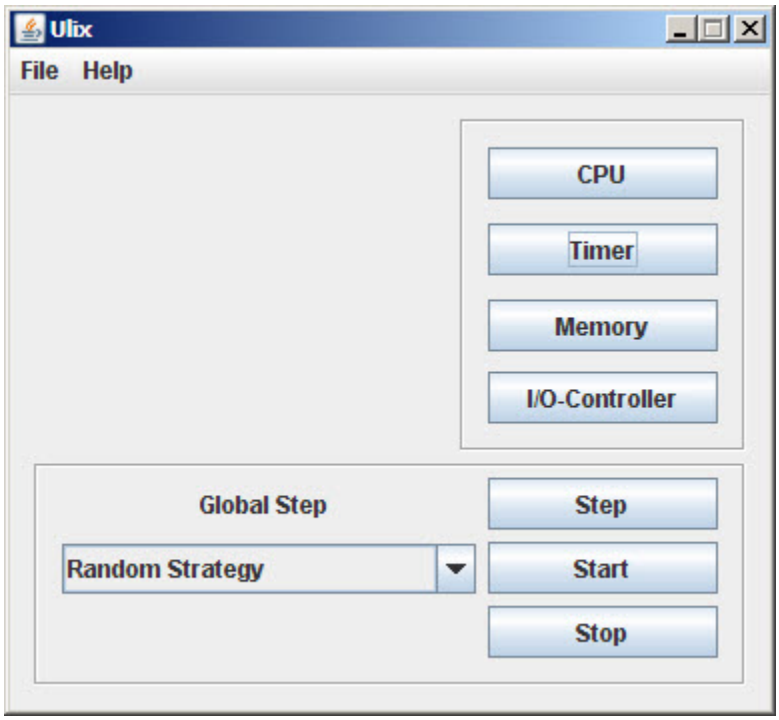

Abbildung 2: Hauptmenü von Ulix

In dem Hauptmenü befinden sich 3 kleine Gruppen (siehe Abbildung 2):

- Menüs des Hauptfensters
- Komponenten des Fensters
- Ablaufoptionen

Im Folgenden werde ich die drei Gruppen von den Bedienelementen getrennt vorstellen.

3.3 Menüs des Hauptfensters

Auf diesem Untermenü „File“ kann man 5 Optionen wählen (siehe Abbildung 3).

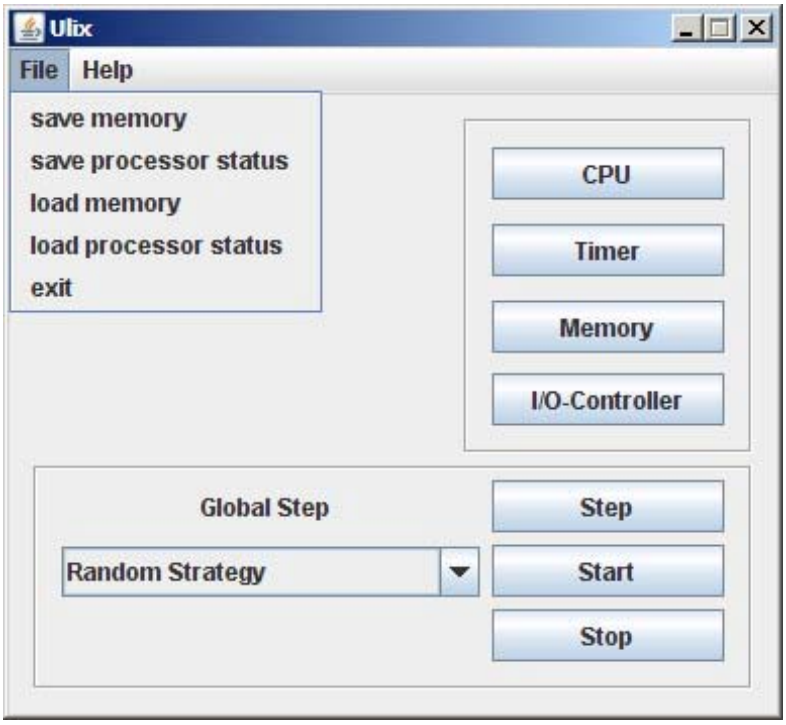

Abbildung 3: Untermenü „File“

1. Save memory: Es werden alle Werte vom Hauptspeicher zu einer Datei gespeichert.
2. Save processor status: Es werden alle Werte, die auf Registern in CPU gespeichert werden, zu einer Datei abgeschrieben.
3. Load memory: Es werden alle Werte von der auszuwählenden Datei zum Hauptspeicher geladen.
4. Load processor status: Es werden alle Werte von der auszuwählenden Datei auf alle Register in CPU abgeschrieben.

3.4 Komponenten-Fenster

Das sind die Button-Gruppen, um den Status der Computer-Teile zu überprüfen und zu ändern (CPU, Timer, Memory, I/O-Controller).

3.4.1 CPU (siehe Abbildung 4)

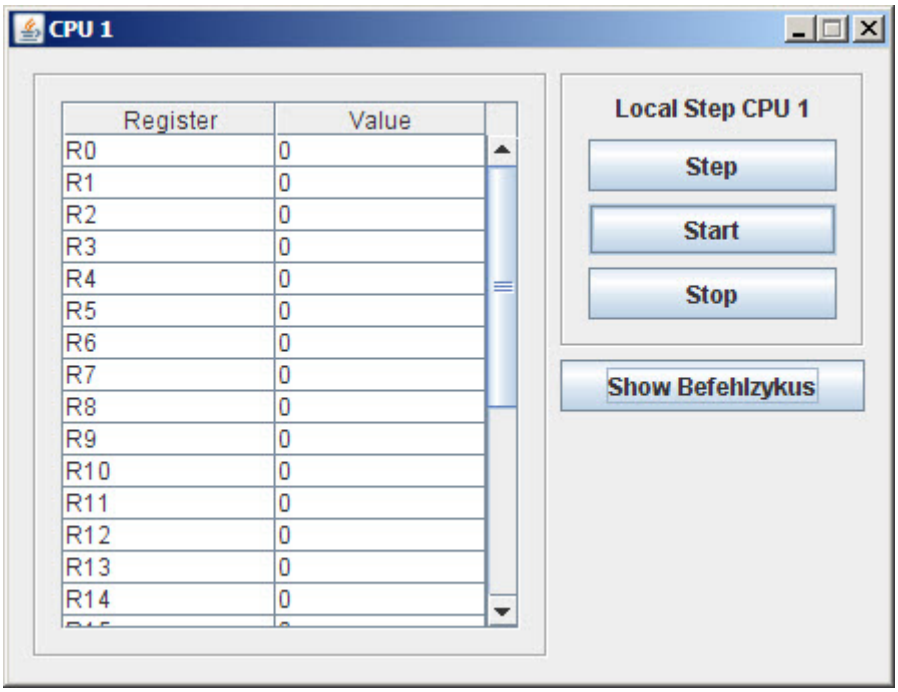

Abbildung 4: CPU

Auf dem Fenster von CPU werden alle Register in der Tabelle dargestellt, und alle Werte werden sofort aktualisiert, wenn die Werte sich verändern. Mann kann den Wert eines Registers auch selbst ändern, indem es auf der Tabelle gedrückt wird und neue Wert eingeben werden, und man kann nicht nur das normale Dezimalzahlensystem eingeben, sondern auch das Oktalzahlensystem, indem man mit dem „0x“ vorher die Nummer (Oder A-F) eingibt.

In dem rechten Fenster befinden sich noch die 4 Buttons „Step“, „Start“, „Stop“ und „Show Befehlzyklus“

- Step: Es wird nur 1 Command Step von dem CPU ausgeführt.
- Start: Es lässt sich Command Step von dem CPU ausführen, bis „Stop“ gedrückt wird.
- Stop: Der ausgeführte CPU-Command Step wird angehalten?
- Show Befehlszyklus (siehe Abbildung 5): Es wird der Befehlszyklus des CPU angezeigt.

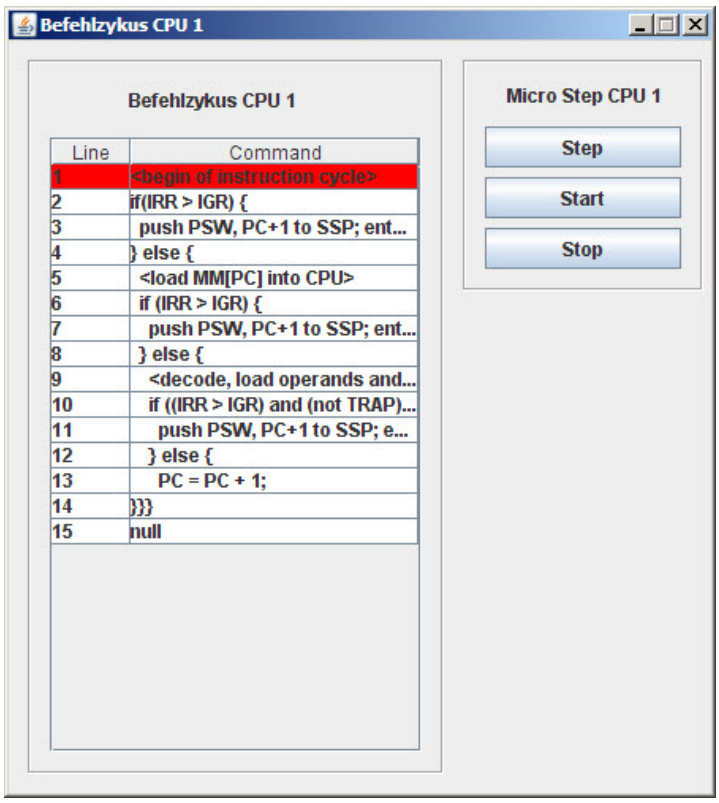

Abbildung 5: Befehlszyklus des CPU

In diesem Fenster lässt sich der Befehlszyklus des CPU in der Tabelle anzeigen, die Tabelle besteht aus Zeilennummer und Steuerbefehlszyklus, die sich gerade auf dem CPU befindet, und es wird mit rotem Hintergrund markiert, um zu wissen, auf welchen Zeilen CPU bearbeitet wird.
Im rechten Fenster stehen noch 3 Optionen zur Verfügung, und zwar „Step“, „Start“ und „Stop“.

- Step: Es wird nur 1 Mikrostep auf dem Steuerbefehlszyklus durchgeführt.
- Start: Mikrostep wird ausgeführi, bis „Stop“ gedrückt wird
- Stop: Der ausgeführte Mikrostep wird angehalten.

3.4.2 Timer (siehe Abbildung 6)

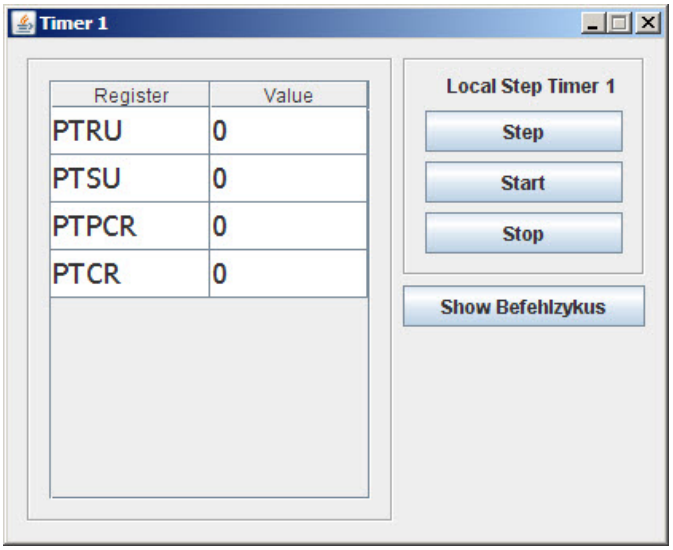

Abbildung 6: Timer

Es werden die Werte der Register PTRU, PTSU, PTPCR und PTCR angezeigt. Man kann die Werte ändern, indem man direkt in der Tabelle neue Werte eingibt. In diesem Fenster befinden sich noch 4 weitere Buttons:

- Step: 1 Step von Command-Befehl in dem Timer wird ausgeführt.
- Start: es lässt sich Command-Befehl auf dem Timer laufen, bis „Stop“ gedrückt wird.
- Stop: Der laufende Command-Befehl in dem Timer wird blockiert.
- Show Befehlszyklus: die Command-Befehl wird in einem neuen Fenster dargestellt (siehe Abbildung 7), und man kann auch den Mikrostep ausführen lassen oder überprüfen.

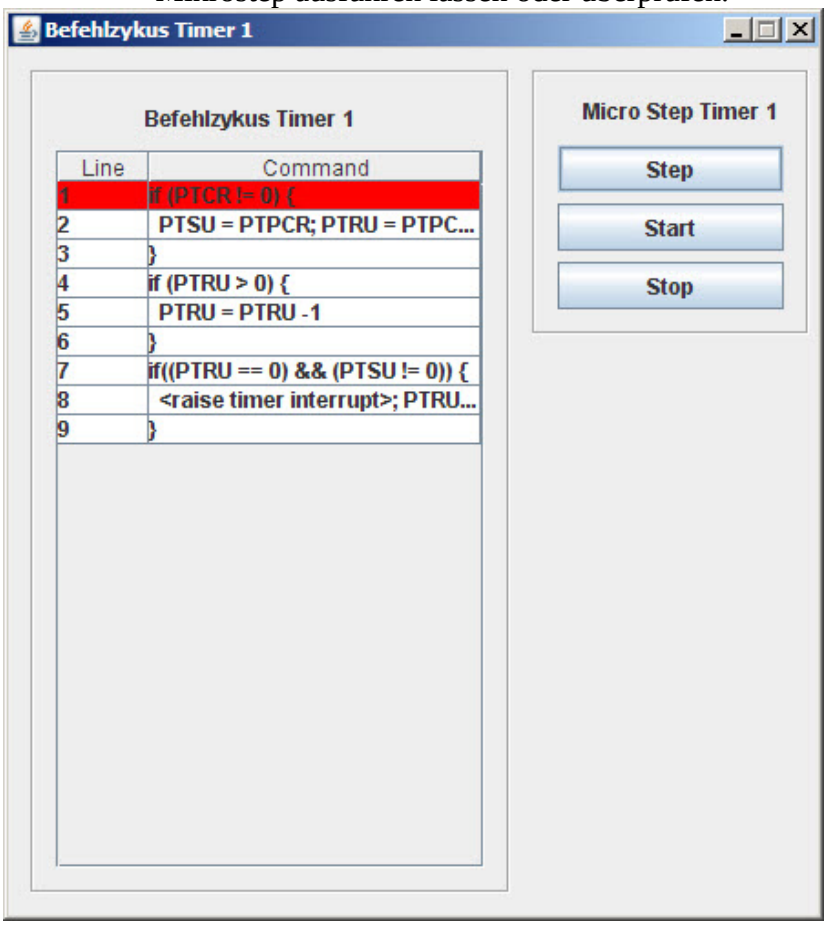

Abbildung 7: Befehlszyklus von Timer

3.4.3 Memory (siehe Abbildung 8)

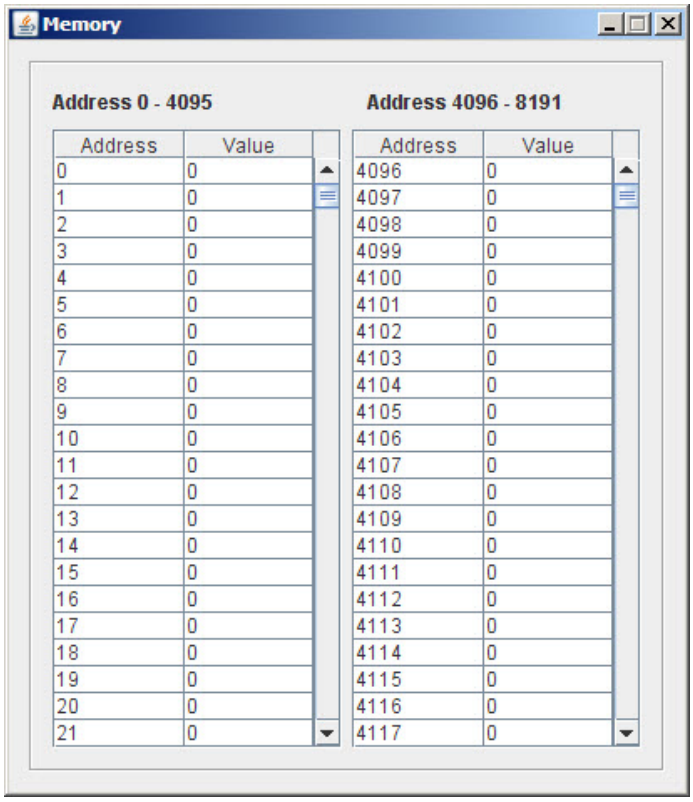

Abbildung 8: Memory

In diesem Fenster werden alle Werte, die im Hauptspeicher gespeichert sind, angezeigt. Es besteht aus 2 Tabellen, die linke Tabelle zeigt die Werte von Adresse 0 bis maximal Adresse/2 und die rechte Tabelle informiert über die Adresse von maximal Adresse/2 bis maximale Adresse, damit man einfach überwachen kann.

3.4.4 I/O-Controller

Auf dem Fenster von I/O-Controller werden die wichtigsten Registers dargestellt, und man kann auch Localstep und Mikrostep von I/O-Controller abspielen (Step, Start, Stop, Show Befehlszyklus). Es ist genau so wie CPU und Timer.

3.5 Ablaufoptionen

Das ist die Button-Gruppe, um Global-Step auszuführen, es besteht aus einer Combo-Box, um die Strategy auszuwählen, und den 3 Buttons Step, Start und Stop.

- Combo-Box (siehe Abbildung 9)

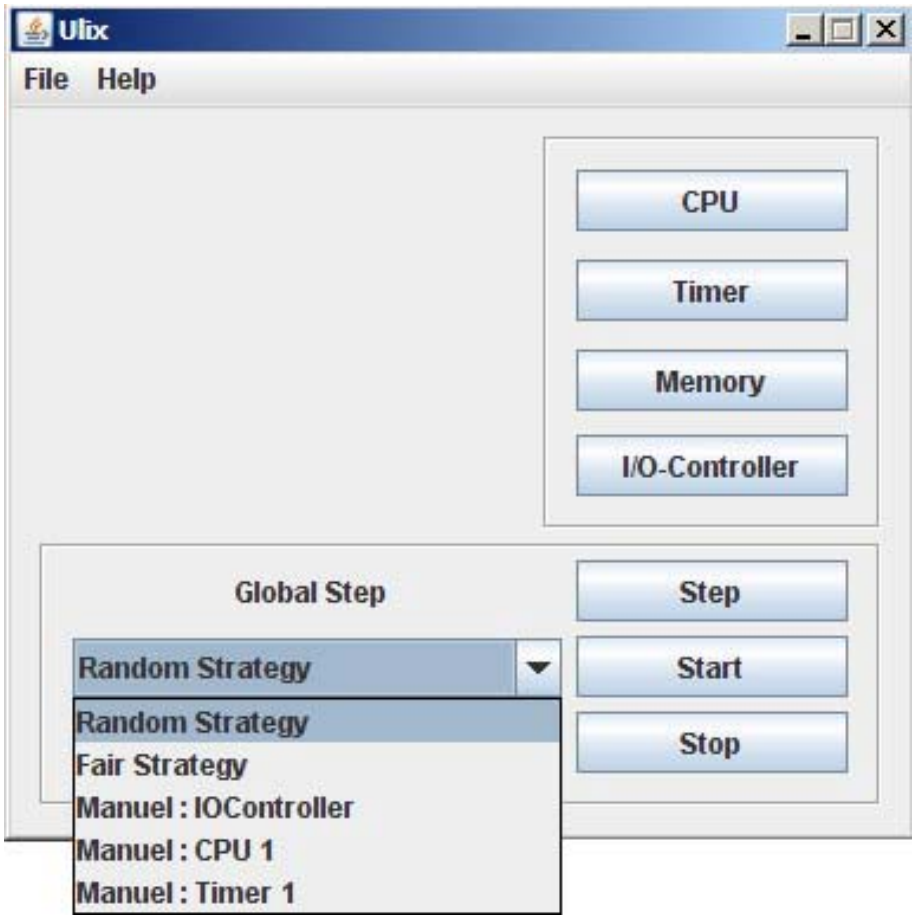

Abbildung 9: Combo-Box

Man kann 2 Hauptstrategien wählen:

- Random Strategy: Alle Komponenten werden zufällig durchgeführt.
- Fair Strategy: Alle Komponenten werden in der Reihe zugeordnet und werden nacheinander durchgeführt.

Man kann auch Localstep von allen Komponenten in diesem Fenster ausführen lassen, indem man eine Komponente von Combo-Box wählt.

- Step: Es wird nur 1 Step von Command-Befehl einer Komponente, die ausgewählt von Combo-Box ist, durchgeführt.
- Start: Es lässt sich Command-Befehl einer Komponente, die von Combo-Box ausgewählt wurde, ausführen, bis „Stop“ gedrückt ist.
- Stop: Der laufende Command-Befehl wird angehalten.

3.6 Zusammenfassung

Dieses Kapitel hat sich damit befasst, wie man Ulix-GUI startet und kontrolliert. Im nächten Kapital wird der Entwurf des Ulix GUI beschrieben.

4. Entwurf des Ulix-GUI

4.1 Einleitung

In diesem Kapital werde ich erklären, wie ich Ulix-GUI entworfen habe, wie viele Klassen es in Ulix-GUI gibt und was die wesentlichen Operationen bei allen Klassen sind.

4.2 Überblick über den Entwurf

Model-View-Controller (MVC, „Modell/Präsentation/Steuerung“) bezeichnet ein Architekturmuster zur Strukturierung der Software-Entwicklung in die drei Einheiten: *Datenmodell* (engl. *Model*), *Präsentation* (engl. *View*) und *Programmsteuerung* (engl. *Controller*). Ziel des Musters ist es, einen flexiblen Programmentwurf zu machen, der u. a. eine spätere Änderung oder Erweiterung erleichtert und eine Wiederverwendbarkeit der einzelnen Komponenten ermöglicht [1]. Hier wird die Präsentation implementiert, und sie besteht aus 6 Klassen: MainPage.java, CpuGUI.java, TimerGUI.java, IOControllerGUI.java, MemoryGUI.java und FunctionsGUI.java (siehe Abbildung 10).

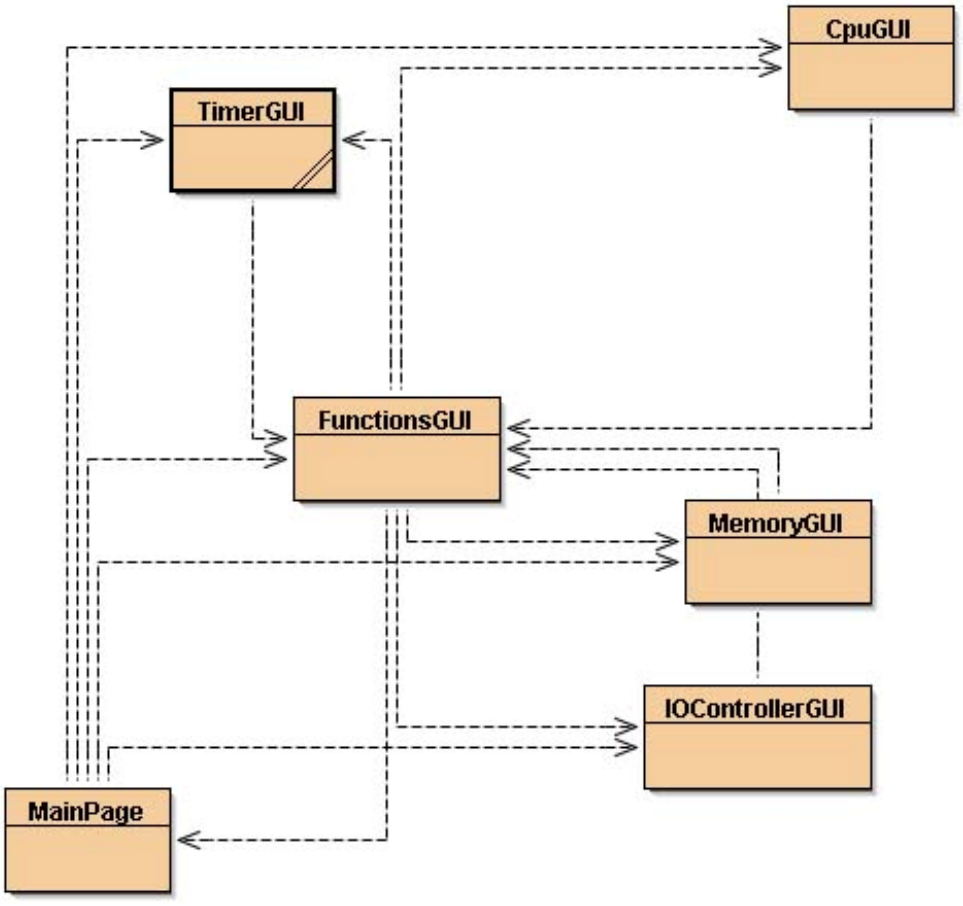

Abbildung 10: Ulix-GUI

Wenn das Ulix-Gui gestartet ist, liest zuerst MainPage.java alle Informationen von der Configdatei und setzt alle Parameter im Ulix-Gui um.
Dies wird von Configdatei gelesen.

- Anzahl Prozessoren
- Größe des Hauptspeichers
- Größe des Sektors
- Größe der Disk

Die Standardeinstellung von allen Parametern lauten:

CPU = 1
Memory = 8192
Disk Size = 100
Sector Size = 512

4.3 MainPage.java

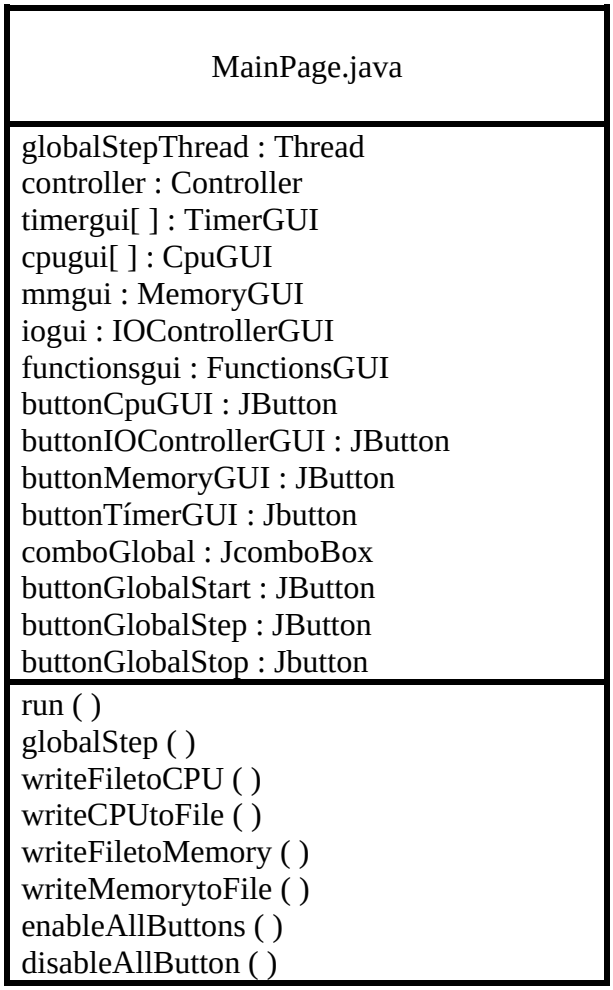

Abbildung 11: Die vollständige Schnittstelle der Klasse „MainPage.java“

Diese Klasse ist das Hauptfenster von Ulix-GUI; auf dieser Klasse werden alle Komponenten verbunden, und Globalstep befindet sich in dieser Klasse. Die vollständige Signatur der Klasse ist in Abbildung 11 zu sehen und die wichtigsten Attribute der Klasse sind:

- globalStepThread : Thread von Globalstep
- buttonCpuGUI : Button, um Informationen von CPU anzuzeigen
- buttonTimerGUI : Button, um Informationen von Timer anzuzeigen
- buttonIOControllerGUI : Button, um Informationen von IOController anzuzeigen
- buttonMemoryGUI : Button, um Hauptspeicher anzuzeigen
- comboGlobal : Combo-Box, um Optionen für Globalstep zu wählen
- buttonGlobalStart : Button für Start (Globalstep)
- butonGlobalStop : Button für Stop (Globalstep)

Die wichtigsten Operationen der Klasse sind:

- run( )
  Overridding Methode von Runnable, damit globalStepThread(Thread) sich ausführen lassen kann, bis „Stop“ gedrückt wird (globalStepThread = null)
- globalStep( )
  Operation, um die Methode auf Controller.java aufzurufen, danach wird eine Komponente durchgeführt.
- writeFiletoCPU( )
  Alle Werte der Register, die auf einer Datei gespeichert sind, werden zu CPU abgeschrieben.
- writeCPUtoFile( )
  Alle Werte der Register werden zu einer Datei abgeschrieben.
- writeFileToMemory( )
  Alle Werte werden von einer Datei auf Hauptspeicher abgeschrieben.
- writeMemoryTOFile( )
  Alle Werte, die auf dem Hauptspeicher gespeichert sind, werden zu einer Datei abgeschrieben.
- enableAllButtons( )
  Alle Buttons (Step, Start, Stop) werden freigegeben.
- disableAllButton( )
  Alle Buttons (Step, Start, Stop) werden blockiert.

4.4 CpuGUI.java

CpuGUI.java
cpuLocalStepThread : Thread cpuMikroStepThread : Thread selectThread : int cpu : CPU functionsgui : FunctionsGUI cpuCount : int currentCount : int localCommands[ ] : String modelTableCpuBefehl : DefaultTableModel modelTableCpuRegister : DefaultTableModel buttonBefehlzykus : JButton buttonCpuLocalStart : JButton buttonCpuLocalStep : JButton buttonCpuLocalStop : JButton buttonCpuMicroStart : JButton buttonCpuMicroStep : JButton buttonCpuMicroStop : JButton tableCpuBefehl : JTable tableCpuRegister JTable
run ( ) cpuLocalStep ( ) cpuMicroStep ( ) refreshTableCpuRegister ( ) refreshTableCpuBefehl ( ) markRow ( ) tableCpuRegisterPropertyChange(evt : PropertyChangeEvent) tableCpuRegisterChange ( ) enableAllButtons ( ) disableAllButtons ( )

Abbildung 12: Die vollständige Schnittstelle der Klasse „CpuGUI.java“

In diesem Fenster befinden sich alle Informationen von CPU (es wird in der Klasse CPU.java aufgerufen, und wird an CpuGUI weitergeleitet) und man kann Localstep von dem CPU ausführen und überprüfen lassen.

Die wichtigsten Attribute der Klasse sind:

- cpuLocalStepThread : Thread von Localstep (CPU)
- cpuMikroStepThread : Thread von MikroStep (CPU)
- cpuCount : Wie viele Prozessoren hat das Emulatorsystem? (cpuCount wird von Controller.java über MainPage.java weitergeleitet)
- cpu : Bezieht sich auf CPU.java, um alle Informationen auf dem CPU zu erhalten
- localCommands[ ] : Bezieht sich auf den Befehlszyklus von CPU
- modelTableCpuBefehl : Wird gebraucht, um die Befehlszyklustabelle bearbeiten zu können.
- modelTableCpuRegister : Wird gebraucht, um Registertabelle bearbeiten zu können.
- tableCpuBefehl : Befehlszyklus wird auf dieser Tabelle angezeigt (durch modelTableCpuBefehl).
- tableCpuRegister : Alle Register werden in dieser Tabelle dargestellt. (durch modelTableCpuBefehl)

Die wichtigsten Operationen der Klasse sind:

- run ( )
  Overridding Methode von Runnable, damit cpuLocalStepThread oder cpuMikroStepThread ausgeführt werden kann, bis „Stop“ gedrückt wird, dann werden cpuLocalStepThread oder cpuMikroStepThread null gesetzt.
- cpuLocalStep( )
  Methode LocalStep( ) (auf CPU.java) wird aufgerufen.
- cpuMikrostep( )
  Methode MikroStep( ) (auf CPU.java) wird aufgerufen.
- refreshTableCpuRegister( )
  alle Registerwerte werden von CPU.java aufgerufen und in dieser Tabelle dargestellt.
- refreshTableCpuBefehl( )
  localCommands[ ] wird durch die Methode getLocalStepRepresentation( ) : String[ ] (auf CPU.java) gesetzt und im tableCpuBefehl angezeigt.
- markRow( )
  Hintergrund im tableCpuBefehl wird markiert, bis die Command-Befehlszeile bearbeitet wird (durch die Methode getMicroStepNummer( ) : int).
- tableCpuRegisterPropertyChange(evt : PropertyChangeEvent)
  kümmert sich darum, wenn ein Wert im tableCpuRegisters geändert wird, danach wird die Methode tableCpuRegisterChange( ) aufgerufen,
- tableCpuRegisterChange( )
  Wenn sich ein Wert auf GUI ändert, dann muss auch der Wert auf CPU.java geändert werden.
- enableAllButtons( )
  Alle Buttons werden blockiert.
- disabkeAllButtons( )
  Alle Buttons werden freigegeben.

4.5 TimerGUI.java

TimerGUI.java
timerLocalStepThread : Thread timerMikroStepThread : Thread selectThread : int timer : Timer functionsgui : FunctionsGUI cpuCount : int currentCount : int localCommands[ ] : String modelTableTimerBefehl : DefaultTableModel modelTableTimerRegister : DefaultTableModel buttonBefehlzykus : JButton buttonTimerLocalStart : JButton buttonTimerLocalStep : JButton buttonTimerLocalStop : JButton buttonTimerMicroStart : JButton buttonTimerMicroStep : JButton buttonTimerMicroStop : Jbutton tableTimerBefehl : JTable tableTimerRegister JTable
run ( ) timerLocalStep ( ) timerMicroStep ( ) refreshTableTimerRegister ( ) refreshTableTimerBefehl ( ) markRow ( ) tableTimerRegisterPropertyChange(evt : PropertyChangeEvent) tableTimerRegisterChange ( ) enableAllButtons ( ) disableAllButtons ( )

Abbildung 13: Die vollständige Schnittstelle der Klasse „TimerGUI.java“

Die wichtigsten Attribute der Klasse sind:

- cpuCount : Wie viele Prozessoren hat das Emulatorsystem? Wenn es 2 CPU gibt, dann gibt es auch 2 Timer.
- timerLocalStepThread : Thread von Localstep (Timer)
- timerMikroStepThread : Thread von Mikrostep (Timer)
- localCommands[ ] : Bezieht sich auf Befehlszyklus des Timer.
- modelTableTimerBefehl : Wird gebraucht, um Befehlszyklustabelle bearbeiten zu können.
- modelTableTimerRegister : Wird gebraucht, um Registertabelle bearbeiten zu können.
- timer : Bezieht sich auf Timer.java, um alle Informationen auf dem Timer zu erhalten.
- tableTimerBefehl : Befehlszyklus wird auf dieser Tabelle angezeigt (durch modelTableTimerBefehl).

- tableTimerRegister : alle Register werden in dieser Tabelle dargestellt. (durch modelTableTimerRegister).

Die wichtigsten Operationen der Klasse sind:

- run ( )
  Overridding Methode von Runnable, damit timerLocalStepThread oder timerMikroStepThread sich ausführen lassen kann, bis „Stop“ gedrückt wird, dann werden timerLocalStepThread oder timerMikroStepThread null gesetzt.
- timerLocalStep( )
  Methode LocalStep( ) (auf Timer.java) wird aufgerufen.
- timerMikrostep( )
  Methode MikroStep( ) (auf Timer.java) wird aufgerufen.
- refreshTableTimerRegister( )
  alle Registerwerte werden von Timer.java aufgerufen und in dieser Tabelle dargestellt.
- refreshTableTimerBefehl( )
  localCommands[ ] wird durch die Methode getLocalStepRepresentation( ) : String[ ] (auf Timer.java) gesetzt und im tableTimerBefehl angezeigt.
- markRow( )
  Hintergrung im tableTimerBefehl wird markiert bis die Command-Befehlszeile bearbeitet wird (durch Methode getMicroStepNummer( ) :int).
- tableTimerRegisterPropertyChange(evt : PropertyChangeEvent)
  kümmert sich darum, wenn sich ein Wert im tableTimerRegister ändert,
  danach wird die Methode tableTimerRegisterChange( ) aufgerufen,
- tableTimerRegisterChange( )
  Wenn einen Wert auf Gui geändert wird, dann muss den Wert in Timer.java auch geändert werden.
- enableAllButtons( )
  Alle Buttons werden blockiert.
- disabkeAllButtons( )
  Alle Buttons werden freigegeben.

4.6 IOControllerGUI.java

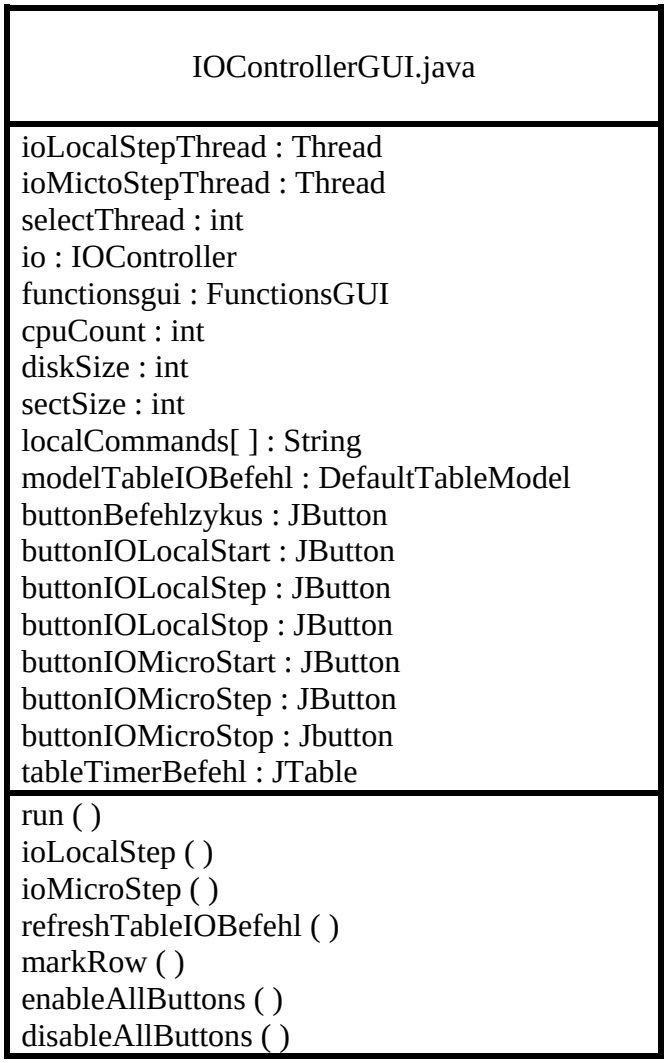

Abbildung 14: Die vollständige Schnittstelle der Klasse „IOControllerGUI.java“

Die wichtigsten Attribute der Klasse sind:

- siehe CpuGUI.java und TimerGUI.java

Die wichtigsten Operationen der Klasse sind:

- siehe CpuGUI.java und TimerGUI.java

4.7 MemoryGUI.java

MemoryGUI.java
cpuCount : int memorySize : int Memory : Memory Functionsgui : FunctionsGUI modelTable1 : DefaultTableModel modelTable2 : DefaultTableModel m_memory[ ] : byte m_mmioName[ ] : String table1 : JTable table2 : JTable
memoryRefresh ( ) refreshTable1 ( ) refreshTable2 ( ) table1PropertyChange (evt : PropertyChangeEvent) table1Change ( ) table1PropertyChange (evt : PropertyChangeEvent) table2Change ( )

Abbildung 15: Die vollständige Schnittstelle der Klasse „MemoryGUI.java“

Die wichtigsten Attribute der Klasse sind:

- memorySize : Wie groß ist der Hauptspeicher?
- Memory : Bezieht sich auf Memory.java.
- modelTable1 : Zeigt Bearbeitung der Tabelle1.
  (Adresse 0 bis maximale Adresse/2)
- modelTable2 : Zeigt Bearbeitung der Tabelle2.
  : (maximale Adresse/2 bis maximale Adresse)
- table1 : Werte von Adresse 0 bis maximale Adress/2 werden in dieser Tabelle dargestellt (durch modelTable1).
- table2 : Werte von maximale Adresse/2 bis maximale Adresse werden in dieser Tabelle dargestellt (durch modelTable1).

Die wichtigsten Operationen der Klasse sind:

- table1PropertyChange (evt : PropertyChangeEvent)
  kümmert sich darum, wenn ein Wert in table1 geändert wird,
  danach wird Methode table1Change( ) aufgerufen,
- table1Change( )
  geänderte Werte werden zu Memory.java weitergeleitet und überschieben
- table2PropertyChange (evt : PropertyChangeEvent)
  kümmert sich darum, wenn ein Wert in table2 geändert wird,
  danach wird Methode table2Change( ) aufgerufen,
- table2Change( )
  geänderte Werte werden zu Memory.java weitergeleitet und überschieben
- refreshTable1( )
  es wird Memory.java aufgerufen, um alle Memorywerte zu erhalten.
- refreshTable2( )
  es wird Memory.java aufgerufen, um alle Memorywerte zu erhalten.
- memoryRefresh( )
  Methode refreshTable1 und refreshTable2 werden aufgerufen.

4.8 FunctionsGUI.java

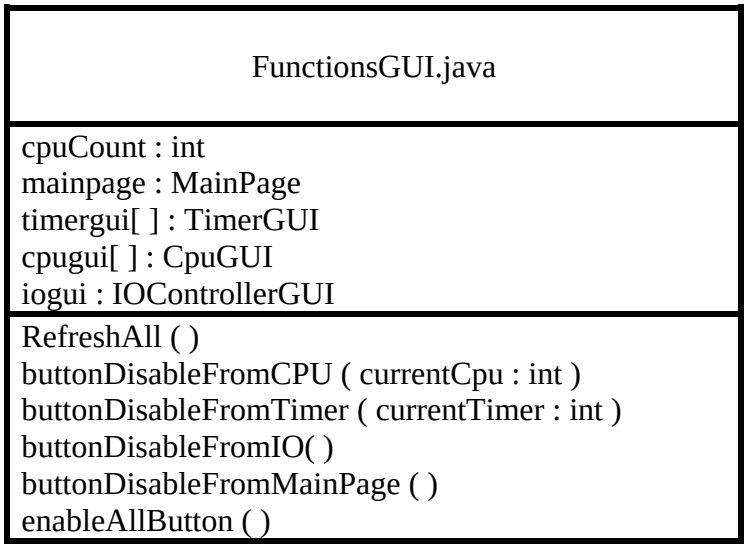

Abbildung 16: Die vollständige Schnittstelle der Klasse „FunctionsGUI.java“

Diese Klasse existiert nur für Button Controller, d.h. alle Buttons werden von dieser Klasse blockiert, während es noch einen laufenden Step gibt, weil es nur maximal 1 ausgeführten Step erlaubt ausgeführt zu werden. Bei dieser Klasse gibt es noch die wichtige Methode RefreshAll ( ). Diese Methode wird jedes Mal aufgerufen, um alle Tabellen in Ulix-GUI zu aktualisieren, d.h. es muss nicht über alle Klassen zugreifen, sondern nur über 1 Methode auf dieser Klasse.

Die wichtigsten Attribute der Klasse sind:

- mainpage : Objekt von MainPage.java
- timergui : Objekt von TimerGUI.java
- cpugui : Objekt von CpuGUI.java
- iogui : Objekt von IOControllerGUI.java

Die wichtigsten Operationen der Klasse sind:

- RefreshAll( )
  Alle Tabellen in Ulix-GUI werden aktualisiert.
- buttonDisableFromCPU( )
  Wenn Start in dem Fenster CpuGUI gedrückt wird, wird diese Methode aufgerufen, um alle Buttons zu blockieren (außer Stop-Button in cpuGUI).
- buttonDisableFromTimer( )
  Wenn Start in dem Fenster TimerGUI gedrückt wird, wird diese Methode aufgerufen, um alle Buttons zu blockieren (außer Stop-Button auf TimerGUI).
- buttonDisableFromIO( )
  Wenn Start in dem Fenster IOControllerGUI gedrückt wird, wird diese Methode aufgerufen, um alle Buttons zu blockieren (außer Stop-Button auf IOControllerGUI).
- buttonDisableFromMainPage ( ).
  Wenn Start auf dem Fenster MainPage gedrückt wird, wird diese Methode aufgerufen, um alle Buttons zu blockieren (außer Stop-Button auf MainPage).

4.9 Zusammenfassung

In diesem Kapital habe ich verdeutlicht, wie ich Ulix-GUI entworfen habe. Im folgenden Kapitel werde ich erklären, wie ich Ulix-GUI implementiert und welche Tools ich benutzt habe.

5. Implementierung des Ulix GUI

5.1 Einleitung

In diesem Kapital werde ich erklären, wie ich das Ulix-GUI implementiert, welche Tools ich benutzt und welche interessanten Problemstellungen ich während der Implementierung festgestellt habe.

5.2 NetBean IDE

Ulix GUI wurde mit Java-Sprache und mit Hilfe der Integrierten Entwicklungsumgebung (engl. Integrated development environment) NetBeans IDE 6.0.1 geschrieben.

NetBeans IDE ist eine Entwicklungsumgebung, die komplett in der Programmiersprache Java geschrieben wurde und auf der NetBeans Plattform läuft. NetBeans IDE wurde hauptsächlich für die Programmiersprache Java entwickelt. NetBeans ist ein Open-Source-Projekt, welches als Plattform für eigene Anwendungen verwendet werden kann. Mit einem sehr großen Nutzerkreis, einer ständig wachsenden Community und über 100 Partnern weltweit ist NewBeans eine der führenden integrierten Entwicklungsumgebungen[2].

NetBeans IDE kann bei dem Entwurf des Fensters und der Eventerstellung helfen, indem man Toolbar von Netbeans auswählt.

5.2.1 Projekterstellung durch Netbean

Man kann eine neue Java-Application durch Netbeans erstellen, indem man auf das Menü File klickt und New Project und „Java Application“ auswählt (siehe Abbildung 17).

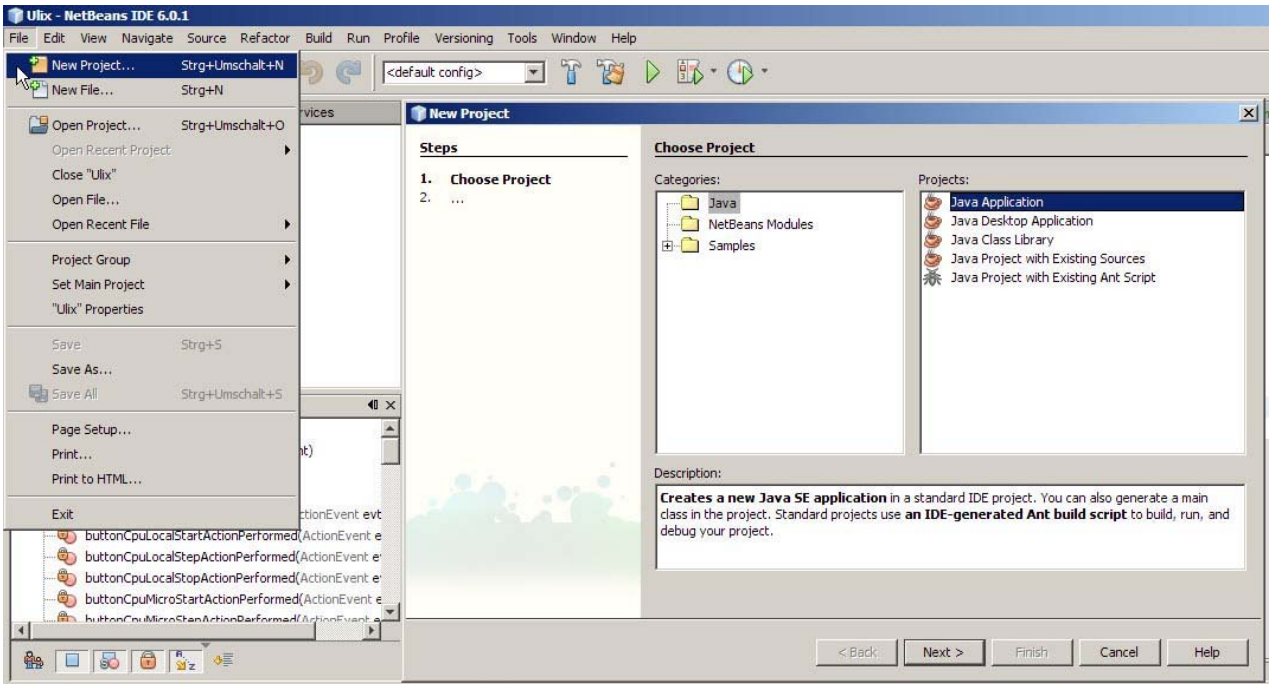

Abbildung 17: Projekterstellung durch Netbean

5.2.2 Eventerstellung durch Netbean

Auf dem JFrame man kann verschiedene Buttons (z.B. Label, Button, Check Box) unter Menu „Swing Controls“ auswählen, und unter dem ausgewählten Button ist es möglich, ein Event zu erstellen, indem man „Events“ drückt (siehe Abbildung 18). Wenn ein Event ausgewählt wird, wird automatisch die Operation auf den Quellcode geschrieben, und man kann auf diese Operation einen Befehl programmieren, um etwas zu befehlen.

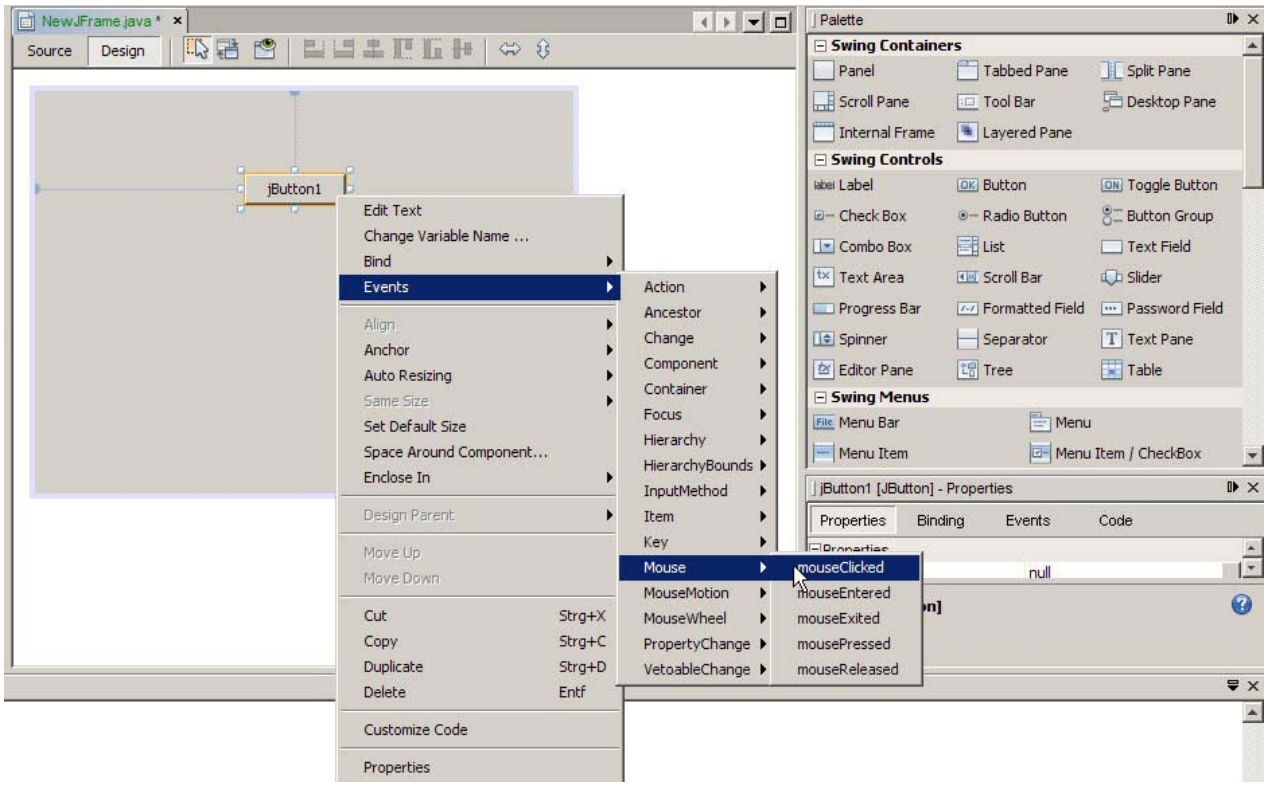

Abbildung 18: Eventerstellung durch Netbean

Zb. Auf der Schaltfläche jButton1 wird das Event „mouseClicked“ gedrückt, dann wird diese Operation automatisch auf den Quellcode hinzugefügt (siehe Abbildung 19).

```
private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
}
```

Abbildung 19: Automatische Operation durch NetBeans

Dann kann man in diese Operation einen Programmbefehl schreiben.

NetBeans IDE mit Subversion

Subversion wird auch von NetBeans IDE unterstützt. Diese Erweiterung kann man unter http://subversion.netbeans.org/ herunterladen.

5.3 Interessante Problemstellungen

5.3.1 Objektidentifizierung

Nehmen wir an, wir erstellen 3 Objekte der Klassen A, B und C. Das Problem liegt darin, dass Klasse A die Objekte b und c erstellt, wobei
Objekt b = B und
Objekt c = C entspricht. In Klasse B wird wiederum ein Objekt c erstellt, das aber kein Duplikat von Klasse C ist, obwohl Objekt c in Klasse A als auch Objekt c in Klasse B eine gemeinsame Klasse haben sollten, die sich aber wiederum nicht miteinander die gleiche Klasse teilen.

Lösung: In Klasse A erstellen wir Objekt b und c, danach übergibt man durch Konsturtor in Klasse B die Parameter des Objekts c von Klasse A (siehe Abbildung 20). Die Klasse A, B und C sind in der Abbildung 21 zu sehen.

```
1  public class A
2  {
3      B b;
4      C c;
5
6      public A(){
7          b = new B(this,c);
8          c = new C(this,b);
9      }
10 }
```

```
1  public class B
2  {
3      A a;
4      C c;
5      public B(A a,C c){
6          this.a = a;
7          this.c = c;
8      }
9  }
```

```
1  public class C
2  {
3      A a;
4      B b;
5      public C(A a,B b){
6          this.a = a;
7          this.b = b;
8      }
9  }
```

Abbildung 20: Beispiellösung einer Objektidentifizierung

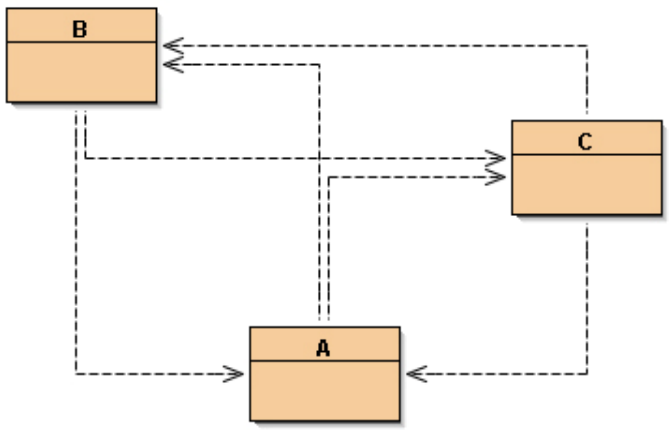

Abbildung 21: Graphische Darstellung der Klassen A,B und C

5.3.2 Thread

Dieses Problem besteht, wenn sich das Ulix-GUI durch den Button „Start“ ausführen lässt, und durch den Button „Stop“ angehalten wird, d.h. während des laufenden Command-Befehls werden die Werte, die auf verschiedenen Tabellen angezeigt sind, aktualisiert, und inzwischen muss das Ulix-GUI auch den Button „Stop“ überwachen.

Lösung: Dieses Problem kann man mit Thread lösen, indem das Thread die Aufgabe der Überwachung des Buttons übernimmt (siehe Abbildung 22).

```
1  Taste „Start“ drücken( ){
2
3      globalStepThread = new Thread(this);
4      globalStepThread.start( );
5
6  }
7  Taste „Stop“ drücken( ){
8
9      globalStepThread = null;
10
11 }
12 public void run( ){
13
14     while(Thread.currentThread( ) == globalStepThread) {
15     globalStep( );
16     }
17
18 }
19
20 public void globalStep( ){
21
22     //Kommandsbefehl1
23     //Kommandsbefehl2
24
25 }
```

Abbildung 22: Thread-Operation

5.3.3 RefreshTable

In jeder Tabelle werden die Werte des Registers dargestellt. Beispielsweise werden bei der Klasse „MemoryGUI.java“ alle Memorywerte in 2 Tabellen angezeigt, da es eine Operation gibt, um die Memorywerte in der Tabelle anzuzeigen (in der Klasse „MemoryGUI.java“ wird die Operation „memoryRefresh( )“ aufgerufen).
Das Problem besteht darin, wenn eine Memorywerte geändert ist, wann und wie wird die Operation „memoryRefesh( ) aufgerufen, weil es mehrere Tabellen auf dem Ulix-GUI gibt und nicht jeder Zeit alle Operationen „Refresh“ auf alle Klassen zugreifen können.

Lösung: Ich habe eine Operation in der Klasse „FunctionsGUI.java“ implementiert, sie heißt refreshAll( ), wenn diese Operation aufgerufen wird, werden alle Werte in allen Tabellen aktualisiert. Und diese Methode wird aufgerufen, wenn sich ein „Step“ ausführen lässt (auch beim Mikrostep, Localstep und Globalstep) (siehe Abbildung 22: Zeile 22).

5.3.4 Disable Button und Enable Button

Wenn man den „Start“-Button drückt, dann lässt sich eine Komponente, die man gewählt hat, ausführen, bis der „Stop“-Button gedrückt wird. Während der laufenden Komponente darf nicht auf andere Komponenten zugegriffen werden, indem alle Buttons auf allen Komponenten blockiert werden. Wenn z.b. der Button „Start“ von CpuGUI.java gedrückt wird, dann werden alle Buttons in allen Klassen durch CpuGUI.java blockiert, bis der Button „Stop“ auf CpuGUI.java angeklickt wird. Dann werden alle Buttons in allen Klassen durch CpuGUI.java freigegeben. Das Problem ist, dass CpuGUI.java alle Klassen kennen muss, um nur diese Operation (Button blockieren) durchzuführen, und es gilt auch für alle Klassen, z.B. TimerGUI.java , IOController.java usw.

Lösung: Ich habe eine Methode in functionsGUI.java geschrieben, um alle Buttons zu blockieren, d.h. jede Klasse muss nur functionsGUI.java kennen.

5.4 Zusammenfassung

In diesem Kapital habe ich die Implementierung des Ulix-GUI erklärt und interessante Problemstellungen während der Programmierung behandelt.

6. Zusammenfassung

6.1 Ergebnisse

Ulix-GUI wurde fertig implementiert. Ich habe das Ulix-GUI nicht nur auf Microsoft Windows getestet, sondern auch auf Linux Ubuntu 8.04. Es funktioniert auf beiden Betriebssystemen sehr gut.

6.2 Ausblick

Wenn ich noch Zeit gehabt hätte, hätte ich noch ein Hauptmenü implementiert, d.h. man müsste nicht die Parameteroptionen via Command-Befehl eingeben, sondern das Ulix-GUI würde zuerst starten, und dann könnte man die Parameteroptionen via GUI (in Fensterform) auswählen.

Des Weiteren möchte ich noch das Dualzahlensystem hinzufügen, d.h. dass man dann nicht nur das Dezimalzahlensystem und das Oktalzahlensystem eingeben kann, sondern auch das Dualzahlensystem.

7. Quellenverzeichnis

[1] Wikipedia: Model-View-Controller, http://de.wikipedia.org/wiki/Model_View_Controller, 28.11.2008

[2] Wikipedia: Netbeans IDE, http://de.wikipedia.org/wiki/NetBeans, 05.12.2008

[3] Java Sun: Javax Dokumente, http://java.sun.com/j2se/1.5.0/docs/api/, 01.11.2008

[4] Kogent Solutions, NetBeans 6 in Simple Steps, Dreamtech Press India, June 2008

[5] Adam Myatt, Pro NetBeans IDE 6 Rich Client Platform Edition, Apress U.S.A, February 2008

[6] Voraset Suwannig, Java GUI using NetBeans, Wannig Thailand, Januar 2008

[7] Vorlesung Betriebssystem, Universität Mannheim, Herbstsemester 2007