Studienarbeit zu dem Thema

Erfassung von personenidentifizierenden Mustern bei Tastatureingaben

Torsten Hermes (202110004)

vorgelegt dem
Fachbereich Informatik der Universität Koblenz - Landau, Campus Koblenz

5. Juli 2007

Referent: Prof. Dr. Rüdiger Grimm
Betreuer: Helge Hundacker

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Seit dem 01. November 2005 ist in Deutschland der neue Reisepass erhältlich. Ein wesentliches Merkmal dieses neuen Passes ist die Einbindung von biometrischen Merkmalen um den Besitzer des Dokumentes zu verifizieren[1]. In anderen Bereichen, wie zum Beispiel der Abwicklung von Vielfliegern an einem Flughafen, halten ähnliche biometrisch gestützte Verfahren Einzug. Weitere Anwendungsmöglichkeiten wären die Absicherung des eigenen Arbeitsplatzes gegen den Zugriff unbefugter Personen, die Verfolgung von Straftätern oder die Verifikation eines Benutzers innerhalb des Internets. Der Wunsch nach Sicherheit in vielen Sektoren steigt zunehmend. Ein Weg diese Sicherheit zu bieten ergibt sich aus den Eigenschaften, die einen Menschen selbst als Unikat auszeichnen.
Das Ziel dieser Studienarbeit besteht darin sich das persönliche Verhalten eines Menschen im Umgang mit einer Tastatur zunutze zu machen um eine Aussage treffen zu können, in wie fern eine Benutzereingabe mit einer vorher generierten Vergleichseingabe übereinstimmt. Der Schwerpunkt liegt dabei in der Erstellung eines Programms, welches in der Lage ist verschiedene Parameter während einer Benutzereingabe zu sammeln, auszuwerten und zu sichern, um den entsprechenden Benutzer zu jeder beliebigen Zeit wieder anhand der abgespeicherten Informationen erkennen zu können. Dabei wird darauf geachtet, dass die entstehende Software auf möglichst vielen bestehenden Systemen ohne größere Probleme angewendet werden kann.

1.1. Basismaterial

Der Studienarbeit liegt das Dokument „Keystroke Biometric Recognition Studies on Long-Text Input over the Internet“ zu Grunde, welches an der Pace University, Pleasantville, New York von Mary Villani, Mary Curtin, Giang Ngo, Justin Simone, Huguens St. Fort, Sung-Hyuk Cha und Charles Tappert herausgegeben worden ist. Innerhalb dieses Dokumentes wird sich mit einem Applet befasst, welches dazu verwendet wird biometrische Werte während einer Benutzereingabe zu erfassen und einen Benutzer anhand dieser Daten versucht zu verifizieren[2].

2. Grundlagen

Dieser Abschnitt dient als eine Einführung in die Thematik der Biometrie und der in der Studienarbeit verwendeten Programmiersprache Java. Des Weiteren werden die verwendeten Arbeitsmittel, welche im Verlauf der Studienarbeit zum Einsatz kommen aufgelistet und kurz beschrieben.

2.1. Allgemeine Einführung in die Biometrie und die Tastenerkennung im Besonderen

Die Biometrie (griech. Bio = Leben und Metron = Maß) befasst sich mit der Lehre der Messung an einem Lebewesen. Von besonderem Interesse sind dabei jene Maße, welche jedem Lebewesen als einzigartig gegeben sind und durch die es von anderen unterschieden werden kann. Dabei unterscheidet man grundlegend zwischen zwei verschiedenen biometrischen Merkmalskategorien, den statischen und den dynamischen. Die statischen Eigenschaften sind stark in der genetischen Veranlagung eines Menschen verankert. Die dynamischen Merkmale hingegen fundieren nicht primär auf genetischen Veranlagungen, sondern sind durch den Umgang mit bestimmten Werkzeugen des alltäglichen Lebens geprägt. Vor allem die statischen Merkmale wie der Fingerabdruck oder Retinascanner werden gegenwärtig immer populärer. Die Tabelle 2.1 zeigt eine Auflistung der bekanntesten biometrischen Merkmale. Dabei wird den einzelnen biometrischen Merkmalen ein genotypischer, ein randotypischer und ein konditionierter Faktor zugewiesen. Die statischen Merkmale liegen dabei in den stark genotypisch und randotypisch veranlagten Merkmalen. Die Merkmale, bei denen ein konditionierter Faktor eine prägnante Rolle spielt sind bei den dynamischen Merkmalen einzuordnen. Dabei bedeutet genotypisch, dass das entsprechende Merkmal in den Genen veranlagt ist, während der randotypische Faktor auf zufälligen Ereignissen beruht, welche in der frühen Entwicklungsphase eines Menschen auftreten. Der koniditionierte Faktor hingegen beruht auf erlernten und an-

trainierten Fähigkeiten[3].

Biometrisches Merkmal	*genotypisch**	*randotypisch**	*konditioniert***
Fingerprint (nur Minuzien)	o	ooo	o
Unterschrift (dynamisch)	oo	o	ooo
Gesichtsgeometrie	ooo	o	o
Irismuster	o	ooo	o
Retina (Blutgefäßstruktur)	o	ooo	o
Handgeometrie	ooo	o	o
Fingergeometrie	ooo	o	o
Venenstruktur der Handrückseite	o	ooo	o
Ohrform	ooo	o	o
Stimme (Klang)	ooo	o	oo
DNA	ooo	o	o
Geruch	ooo	o	o
Tastenanschlag	o	o	ooo
Vergleich: Passwort			(ooo)

Abbildung 2.1.: Tabelle mit den bekanntesten biometrischen Merkmalen Quelle: [3]

Diese Studienarbeit befasst sich, wie schon in der Einleitung erwähnt mit der Erfassung von biometrischen Werten anhand des individuellen Verhaltens einer Person im Umgang mit einer Tastatur. Die Tabelle 2.1 zeigt deutlich, dass diese Art biometrischer Merkmale zu den dynamischen Erkennungsmerkmalen gehört. Dies liegt daran, dass das Tippverhalten eines Menschen nicht stark von seinen Genen beeinflussen wird, es sei denn er hat genetisch bedingte Defizite, die die Handhabung einer Tastatur beeinflussen. Viel mehr handelt es sich hier um eine antrainierte Eigenschaft, die sich mit der Zeit und durch den Umgang mit einem Computer oder einer Schreibmaschine entwickelt. Ein großes Problem, mit dem nicht nur das Tippverhalten, sondern auch alle anderen dynamischen Merkmale zu kämpfen haben ist die Frage der Konsistenz. Es ist nicht sicher ob sich in ei-

nigen Jahren nicht das Schriftbild an der Tastatur durch die tägliche Anwendung ändert, oder ob sich durch das wiederholte Eingeben einer Passphrase, welche für die Verifizierung angewandt wird nicht eine gewisse Abstumpfung Anhand der routinierten Eingabe einstellt. Des Weiteren schwankt das individuelle Verhalten eines Menschen einhergehend mit seinem Gemüts - und Gesundheitszustand. Das genaue Konsistenzverhalten ist daher nicht eindeutig für ein individuelles Schriftbild erkennbar. Um einen Benutzer trotz dieser möglichen Probleme mit einer ausreichenden Treffsicherheit erkennen zu können werden bei Systemen, die mit der Hilfe von biometrischen Daten arbeiten Schwellenwerte benötigt, da ein Benutzer nie eine einhundertprozentige Übereinstimmung erzielen kann. Im Folgenden wird nun kurz der Vorgang beschrieben, nach welchem jedes System arbeitet, welches biometrische Daten verwendet.
Biometrische Systeme benötigen stets, also auch bei dem Programm, welches im Rahmen der Studienarbeit erstellt worden ist, zwei Phasen, die es zu durchlaufen gilt. In der ersten Phase, der sogenannten Enrollment - Phase werden erste Daten gesammelt. Das System lernt hier den Benutzer kennen, der mit dem System interagiert. Es werden die biometrischen Daten des Benutzers gesammelt und verarbeitet. Auf diese Weise entsteht ein Vergleichsdatensatz, auch Template genannt, welcher alle Daten beinhaltet, die benötigt werden um einen Benutzer verifizieren zu können. Wenn die gesammelten Daten ausgewertet und das Template somit vollständig ist, ist die Enrollment - Phase abgeschlossen. Die Aufgabe der zweiten Phase, auch Matching - oder Arbeitsphase, besteht darin eine Benutzereingabe mit dem dazugehörigen Template zu vergleichen und eine dementsprechende Aussage darüber zu treffen, ob der Benutzer verifiziert werden kann oder nicht.
Bei jedem biometrischen System muss man zwingend die Falschakzeptanzrate (FAR) und die Falschrückweisungsrate (FRR) im Auge behalten. Diese beiden Werte stehen für die Möglichkeit einen Benutzer als korrekt zu erkennen, obwohl das Template nicht von dem gleichen Benutzer stammt, welcher die aktuelle Eingabe generiert hat(FAR), oder diesen zurück zuweisen, obwohl das Template und die Benutzereingabe den gleichen Urheber aufweisen(FRR). Ist einer dieser beiden Werte zu hoch, so muss der Schwellenwert, der entscheidet, ob ein Benutzer erkannt wird oder nicht, korrigiert werden. Dabei geht die Sicherheit selbstverständlich vor. Es ist also wichtiger einen geringen Schwellenwert zu erlangen, als zu hohe Fehlerraten zuzulassen.

2.2. Einführung in Java als verwendete Programmiersprache

Da das Programm, welches im Verlauf der Studienarbeit vorgestellt wird, in der Programmiersprache Java verfasst worden ist, wird diese nun kurz vorgestellt.
Java gehört zu den objektorientierten Programmiersprachen. Sie wurde im Jahre 1992 das erste Mal veröffentlicht und wird bis heute von mehreren Firmen und Personen vertrieben. Dabei ist Java ein eingetragenes Markenzeichen die Firma Sun Microsystems. Bei der Entwicklung von Java wurde großen Wert darauf gelegt Unabhängigkeit von jedem bekannten Betriebssystem zu erreichen, so dass Java Programme von jedem beliebigen Arbeitsplatz aus gestartet werden können. Um dies zu ermöglichen laufen Java - Programme in einer eigenen Umgebung, welche auf dem jeweiligen Betriebssystem installiert werden muss. Diese Umgebung bezeichnet man als die Java Virtual Machine. In Java unterscheidet man im groben zwischen zwei unterschiedlichen Programmarten, den Applikationen und den Applets. Applikationen sind offline arbeitende Programme, welche auf dem lokalen Arbeitsplatz ausgeführt werden. Innerhalb der Studienarbeit wurde allerdings keine Applikation, sondern ein Applet erstellt. Ein Applet dient der Arbeit im Internet. Dementsprechend unterliegen sie innerhalb der Virtual Machine anderen Einstellungen. Ein Applet kann innerhalb eines Internet - Browsers ausgeführt werden, wenn dieser ein entsprechendes Java - Plugin installiert hat. Ansonsten kann ein Applet unter anderem mit dem Java Applet Viewer betrachtet werden. Alle Applets sind aus der Klasse Applet, welches sich in dem Paket java.Applet befindet, abgeleitet. Diese Klasse ist eine konkrete Implementierung der Klasse java.awt, welche innerhalb von Java für die Erstellung von grafischen Benutzeroberflächen benötigt wird. Dadurch erbt ein Applet bereits einige wichtige Fähigkeiten aus den abstrakten Klassen, aus denen ein Applet letztendlich abgeleitet ist. Die Abbildung 2.2 spiegelt die genaue Ableitung der Klasse java.Applet wieder.[4, 5]

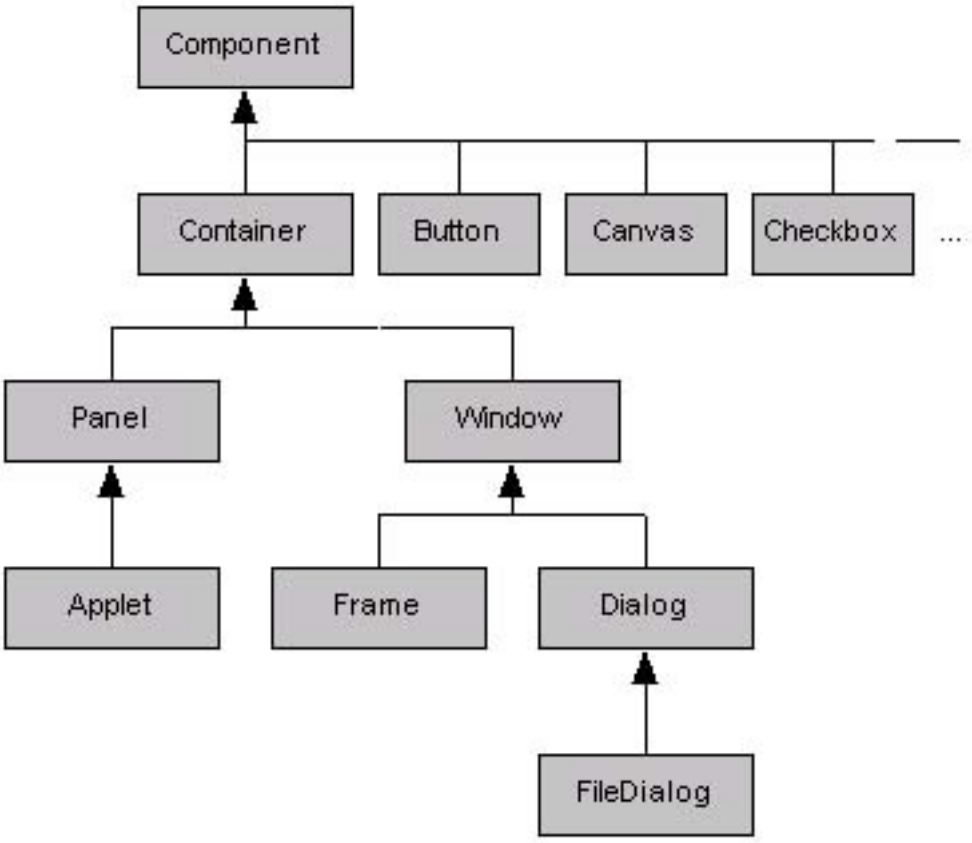

Abbildung 2.2.: Ableitungshierarchie für java.Applet

Wird ein Applet gestartet, so geschieht dies in einem Browser in einem entsprechenden Applet - Tag oder mit Hilfe des Applet Viewers. Dabei wird ein neues Objekt, also eine Instanz der Klasse Applet gebildet. Der Browser initialisiert dann das Applet mit Hilfe der Methode init() und zeigt dann letztendlich das Applet an, welches von dem Browser mit Hilfe der Methode start() gestartet worden ist.

2.3. Verwendete Arbeitsmittel

Um ein Programm erstellen zu können bedarf es verschiedener Werkzeuge. Bei der Arbeit an der Studienarbeit wurden dabei zwei verschiedene Komponenten eingesetzt, welche nun im Folgenden vorgestellt werden.

2.3.1. Java Development Kit

Um überhaupt mit Java als Programmiersprache arbeiten zu können muss ein entsprechendes Softwarepaket der Firma Sun Microsystems[1] auf dem PC installiert werden, das jdk (Java Development Kit). Für das Applet wurde die zu diesem Zeitpunkt aktuelle Version jdk1.5.0_09 verwendet. Dieses Softwarepaket beinhaltet neben den Komponenten der Programmiersprache und deren Kompilierung und Ausführung noch einige Tools, welche zum Teil im Verlauf der Arbeit mit dem Applet noch zum Einsatz kommen. Diese Werkzeuge, welche im Folgenden noch benötigt werden um die Speicherung von Daten auf der Festplatte zu ermöglichen, werden nun im Einzelnen noch kurz vorgestellt. Innerhalb des jdk sind diese Tools unter dem Ordner *bin* aufgeführt.

1. Das erste Tool ist das Porgramm *jar*. Dieses Programm erstellt ausführbare .jar - Archive, welche benötigt werden um das kompilierte Applet in die Lage zu versetzen auf ein persistentes Speichermedium zuzugreifen.

2. Das zweite eingesetzte Programm ist das *keytool*. Dieses wird dazu verwendet ein Schlüsselpaar zu erstellen mit dessen Hilfe ein Zertifikat ausgestellt werden kann.

3. Letztendlich wird noch das jarsigner - tool benötigt um das generierte Zertifikat an das Archiv zu binden.

Die jeweils aktuelle Fassung des jdk mit allen Tools ist im Downloadbereich auf der Internetseite www.sun.com frei erhältlich. Eine ausführliche Erklärung der einzelnen Komponenten befindet sich ebenfalls auf dieser Internetseite im Supportbereich oder wahlweise innerhalb des Ordners *docs* der installierten jdk Distribution.

[1] http://www.sun.com/

2.3.2. Die Entwicklungsumgebung

Um das Applet erstellen zu können benötigt man eine entsprechende Entwicklungsumgebung, welche im Internet in großer Vielzahl, sowohl kostenlos, als auch kostenpflichtig, erhältlich sind. Das Programm, welches im Verlauf der Erstellung des Applets verwendet wurde ist der JCreator in der Version 3.5, welcher von der Firma Xinox Software[2] vertrieben wird. Dieses Programm ist in zwei verschiedenen Fassungen erhältlich. Die eine Fassung, auch als Pro - Fassung bezeichnet ist kostenpflichtig und bietet dementsprechend einige Zusatzfeatures. Die zweite Fassung, welche auch als LE - Fassung bezeichnet wird, bietet alle Basiselemente, welche für den Umgang mit Java benötigt werden und ist kostenfrei. Innerhalb der Studienarbeit wurde die LE - Fassung des JCreators verwendet. Weitere Informationen sowie die Möglichkeit die jeweils aktuelle Fassung der Software herunterzuladen befinden sich auf der angegebenen Homepage.

2.4. Verwandte Projekte

Abschließend wird an dieser Stelle noch kurz ein ähnliches Projekt vorgestellt, welches an der Universität Regensburg entwickelt worden ist und bereits praktische Anwendung findet. Dabei arbeitet das entwickelte Programm *psylock* auf der Basis einer künstlichen Intelligenz. Das System lernt dabei einen Anwender im Lauf der Zeit immer besser kennen und erkennen, wodurch immer genauere Aussagen getroffen werden. Bei der Datenerfassung werden viele verschiedene Parameter gesammelt, welche zum Beispiel auf der Tippgeschwindigkeit, den typischen Fehlern oder der Gewandtheit im Umgang mit den Tasten beruhen. Psylock ist dabei schon so weit fortgeschritten in der Entwicklung, dass es bereits erfolgreich eingesetzt wird, unter anderem bei universitätsinternen Prozessen, wie Passworterneuerungen.[9]

[2] http://www.jcreator.com/

3. Aufbau der grafischen Benutzeroberfläche (GUI)

Wenn man nach der Definition geht, dann ist der Begriff *GUI* die Abkürzung für *Graphical User Interface*. Es ist der technische Begriff für die grafische Benutzeroberfläche[1], über die ein Benutzer mit einem Programm interagiert. Aus diesem Grund lag es auch am nächsten bei dem Aufbau des Applets als erstes die GUI aufzubauen, da im späteren Verlauf über diese nicht nur alle Texteingaben erfolgen, sondern auch Ergebnisse ausgegeben werden. Dieser Abschnitt dient der Vorstellung des Interfaces für das Applet und darüber hinaus einer Erläuterung der Wahl der Java internen Werkzeuge, um dieses aufzubauen.

3.1. Das verwendete Grafikpaket

Am Anfang der Programmierarbeit stand nun also die Aufgabe eine Benutzerschnittstelle zu implementieren, welche funktional, übersichtlich und nicht zu komplex ist. In der bereits im ersten Kapitel erwähnten Arbeit zu dem Thema war eine übersichtliche GUI verwendet worden. Dieses Interface wurde als Basis verwendet um ein ähnliches Interface nachzubilden. Java bietet zu diesem Zweck zwei verschiedene API's an, die Grafikbibliotheken java.awt und javax.swing. Unter der Abkürzung API versteckt sich der Begriff Application Programming Interface. Dabei ist das abstract Window toolkit, kurz AWT, die ältere der beiden Fassungen und Basis für Swing. Swing ist dementsprechend das mächtigere Werkzeug wenn es darum geht eine Oberfläche zu gestalten, allerdings ist es für die Erstellung des Interfaces dieses Applets zu mächtig und zu aufwändig, da bei der Implementierung einer einfach gehaltenen Benutzeroberfläche im Verhältnis zu viel Programmierarbeit zu leisten ist. Ein eher zu vernachlässigendes Problem ergibt sich aus

[1] http://www.informationsarchiv.net/lexikon/26.gui.html

der Version von Java, die eventuell im späteren Verlauf verwendet werden soll. Swing war nicht von Anfang an Bestandteil von Java, sondern wurde erst mit Java Development Kit (jdk) 1.1 als Add - On angeboten. Seit dem JDK 1.2 ist Swing als fester Bestandteil etabliert[2]. Es kann daher vorkommen, dass eventuell ein Rechner mit stark veralteten Installationskomponenten erhebliche Probleme damit hat ein Swing - Interface zu verarbeiten, da es nicht sichergestellt ist die entsprechende Bibliothek auf dem Rechner vorzufinden. Dies ist allerdings ein geringfügiges Problem.
Da bei der Erstellung kein großer Wert auf komplexes grafisches Arbeiten gelegt wird ist also die Verwendung des AWT nicht nur einfacher, sondern auch effizienter. Das AWT bietet, wie sich im weiteren Verlauf noch zeigen wird, eine ausreichende Anzahl an Methoden, um sowohl die Kommunikation zwischen Mensch und Maschine, als auch die Kommunikation zwischen der Oberfläche und den im Hintergrund laufenden Prozessen zu ermöglichen. Es ermöglicht ein zuverlässiges Arbeiten ohne einen zu hohen Aufwand.

3.2. Generierung der GUI

Bei der Erstellung der GUI wurde ein geschachteltes Layout - Management verwendet um die einzelnen Bereiche der Benutzeroberfläche deutlich voneinander abzugrenzen und die Gestaltung dieser effizient und übersichtlich realisieren zu können. Java bietet hierzu die Möglichkeit die Klasse Panel zu verwenden, welche aus der Klasse Container abgeleitet ist und deren Aufgabe darin besteht eine Sammlung von Objekten aufzunehmen, welche zu Dialogzwecken verwendet werden können. Da für den Aufbau der GUI nur Dialogelemente verwendet werden entstehen dementsprechend keine Konflikte bei der Umsetzung der einzelnen Panels. Das Applet wird aus insgesamt vier Panels aufgebaut, welche jeweils eine konkrete Aufgabe verfolgen. Im Folgenden werden nun die einzelnen Panels mit ihren jeweiligen Eigenschaften kurz vorgestellt.

Das Panel p1: Dieses Panel dient nur der Ein- und Ausgabe der Benutzerdaten und besteht ausschließlich aus einer TextArea (siehe Abbildung 3.1, roter Abschnitt). Eine TextArea ist ein mehrzeiliges Eingabefeld. Innerhalb des Applets gibt der Benutzer hier Im Verlauf der Eingabephasen den Text ein und es werden Ergebnisse von einzelnen Auswertungen über die TextArea ausgegeben. Zu diesem Zweck ist

[2] siehe [4], Kapitel 23.1.2

die Area während der Auswertungen für Eingaben durch den jeweiligen Benutzer gesperrt.

Das Panel p2: Dieses Panel beinhaltet vier Textzeilen, sogenannte TextFields, die während der Benutzereingabe einzelne erfasste Werte ausgeben (siehe Abbildung 3.1, blauer Abschnitt). Im Gegensatz zu dem ersten Panel sind hier mehrere Dialogelemente innerhalb eines Panels untergebracht. Zusätzlich ist noch ein entsprechendes Label für jedes einzelne Element beigefügt um anzugeben, welche Daten über das jeweilige Dialogfeld ausgegeben werden.

Das Panel p3: Dieses Panel beinhaltet die Buttons, über die das Applet navigiert wird (siehe Abbildung 3.1, grüner Abschnitt). Im Gegensatz zu den anderen Panels wird hier keine sichtbare Ausgabe getätigt. Dabei dient der linke Button vorwiegend den Löschoperationen, während der rechte den Benutzer durch die einzelnen Phasen des Applets führt. Falls der Benutzer allerdings eine Entscheidung treffen soll, so werden die Buttons entsprechend noch anderweitig angewandt.

Das Panel main: Bis zu diesem Zeitpunkt stehen die einzelnen Panels noch getrennt voneinander innerhalb des Applets. Um die drei Panels mit ihren autarken Layouts zu kombinieren bedarf es noch eines vierten Panels, dem Panel *main*. Die Aufgabe dieses Panels besteht darin, die einzelnen Dialogelemente der drei Panels miteinander zu kombinieren und jedem einzelnen Panel eine Position und ein Größenverhältnis zuzuweisen. Durch die Verwendung dieses vierten Panels spricht man letztendlich von dem oben bereits erwähnten geschachtelten Layout - Management. Das main - Panel behandelt dabei die anderen drei Panels, als wären sie einzelne Dialogelemente und richtet diese zu einem Gesamtbild aus.

Eine genauere Beschreibung des Aufbaus sowie eine Beschreibung der verwendeten Umgebungsvariablen befinden sich innerhalb des Anhangs, Abschnitt A.

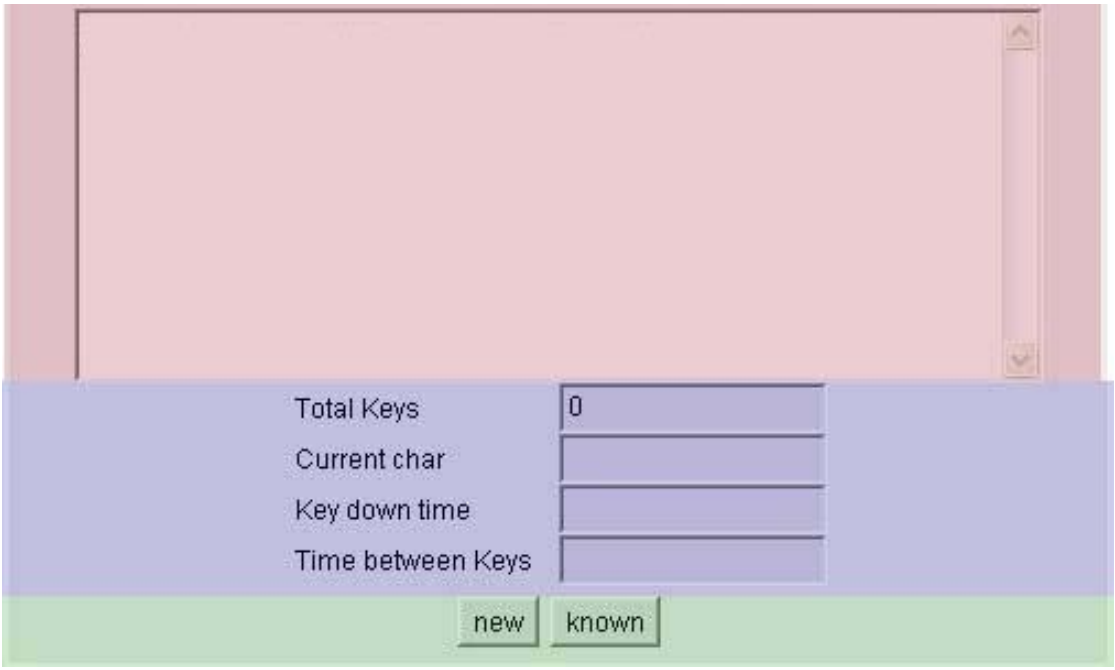

Abbildung 3.1.: Darstellung der GUI

4. Erfassung und Verarbeitung der gewonnenen Daten

Nach der Implementierung der Gui bestand der nächste Arbeitsschritt in der Verarbeitung von Benutzerdaten. In diesem Abschnitt wird nun genauer darauf eingegangen werden, wie diese Verarbeitung von Statten geht und wie die einzelnen erfassten Werte verarbeitet werden.

4.1. Datenerfassung - Die Listener

Um innerhalb des Applets Benutzerdaten erfassen zu können benötigt man Schnittstellen, auch Interfaces genannt, mit deren Hilfe das Programm in der Lage ist die von dem Betriebssystem und der Hardware ausgesendeten Signale korrekt zu erfassen und zu verarbeiten. Java verwendet zu diesem Zweck ein Modell, welches als *Delegation Based Event Handling* Kommunikationsmodell bekannt ist. Dieses Modell basiert auf der Idee von Nachrichten. Dabei kann eine Quelle innerhalb des Systems eine Nachricht aussenden, ein entsprechendes Interface innerhalb eines Java - Programms empfängt dann diese Nachricht und reagiert darauf. Um dabei zu verhindern, dass unkontrolliert alle Arten von Nachrichten empfangen und verarbeitet werden muss man verschiedene Programmkomponenten, wie zum Beispiel einzelne Dialogelemente, registrieren, damit sie für die verschiedenen Arten von Listenern empfänglich sind.
Da jeder Listener in Java als ein Interface, also abstrakt deklariert ist, ist es nötig neben dieser Registrierung die einzelnen Methoden, welche durch eine abstrakte Deklaration innerhalb des jeweiligen Interfaces vorgegeben sind mit einer konkreten Implementierung zu versehen, da das Programm ansonsten nicht lauffähig sein wird.
Innerhalb des Applets werden zwei Listener verwendet um Nachrichten empfangen

und verarbeiten zu können. Dabei handelt es sich um den *KeyListener* und den *ActionListener*, welche nun im Folgenden genauer behandelt werden.

4.1.1. Der KeyListener

Der KeyListener ist von Java dafür vorgesehen worden um Nachrichten empfangen zu können, welche von der Tastatur eines Benutzers durch das Betätigen einer Taste ausgesendet werden. Um die Daten während der Eingabe zu sammeln und später auswerten zu können, ist es zwingend notwendig die Benutzereingabe innerhalb des Applets abfangen zu können. Das KeyListener - Interface bietet diese Möglichkeit, sobald man die TextArea bei dem KeyListener registriert hat. Innerhalb des Interfaces gilt es dann noch insgesamt drei Methoden zu überschreiben, welche dort abstrakt vordefiniert sind und bei verschiedenen Events ausgelöst werden. Diese Methoden lauten:

public void keyPressed: Diese Methode wird genau in dem Moment angesprochen, wenn eine Taste heruntergedrückt worden ist.

public void keyReleased: Diese Methode wird jedes mal ausgeführt, sobald eine Taste wieder gelöst wird.

public void keyTyped: Diese Methode ist eine Kombination aus den beiden vorgenannten Methoden. Sie wird immer dann ausgeführt, wenn eine Taste betätigt worden ist.

Jede dieser drei Methoden wird genau einmal für jede Art eines Tastendrucks innerhalb einer Eingabephase ausgeführt. Innerhalb der einzelnen Eingabephasen, werden durch diese Eigenschaft folgende Funktionen von den KeyListener - Methoden abgedeckt:

1. **Erfassung von Zeitparametern:** Die erste Aufgabe besteht darin verschiedene Zeitstempel mit Hilfe der Java internen Methode *System.currentTimeMillis* abzufragen und daraus die Zeiten zu berechnen, welche der jeweilige Benutzer eine Taste heruntergedrückt hält und die Zeit, welche zwischen dem Lösen einer Taste und dem Herunterdrücken der nächsten Taste verstreicht. Die

daraus gewonnenen Zeitwerte werden dann direkt intern auf einem Array abgespeichert um diese dann bei der Auswertung zu verarbeiten.

2. **Erfassung der einzelnen Zeichen:** Die zweite Aufgabe ist nahe liegend. Da die einzelnen Methoden des KeyListeners immer dann ausgeführt werden, wenn eine Taste betätigt wird, kann man diese Eigenschaft anwenden um die Taste direkt abzuspeichern. Dies geschieht allerdings nicht über das direkte Zeichen, sondern über einen Java internen virtuellen Code, welcher eine entsprechende Taste repräsentiert. Diese Codes werden dann innerhalb des Applets ebenfalls direkt abgespeichert um später darauf zugreifen zu können.

3. **Sicherstellung von korrekten Eingaben:** Durch die Wahl des virtuellen Codes zur Abspeicherung der gewonnen Daten innerhalb des Applets ist sichergestellt, dass keine Befehle einer Programmiersprache abgespeichert werden können. Dadurch ist eine Sicherheit gegenüber dem System gewährleistet und zusätzlich eine Sicherheit gegenüber dem Benutzer gegeben.

4. **Ausgeben von Werten:** Diese Aufgabe ist eher nebensächlich. Innerhalb der einzelnen Methoden werden die errechneten Werte, welche ermittelt worden sind direkt über die dafür vorgesehenen Textzeilen des zweiten Panels ausgegeben um dem Benutzer eine visuelle Resonanz wiedergeben zu können.

5. **Inkrementierung der globalen Arrays:** Die letzte Aufgabe dient der Abspeicherung der einzelnen gewonnenen Werte innerhalb der Methoden. Da zur Abspeicherung der gewonnenen Werte Arrays verwendet werden muss gewährleistet sein, dass die einzelnen Werte gespeichert werden können ohne dabei Überlagerungen an einzelnen Speicherplätzen aufkommen zu lassen. Da die Methoden bei jedem Tastendruck ausgeführt werden ist es naheliegend die Laufvariable für die einzelnen Speicherarrays innerhalb einer dieser Methoden zu inkrementieren, so dass es zu keiner Überlagerung kommen kann.

4.1.2. Der ActionListener

Der zweite Listener, der innerhalb des Applets angewendet wird ist der ActionListener. Im Gegensatz zu dem KeyListener erwartet der ActionListener eine besondere Eingabe, welche von einem Dialogelement ausgesendet wird. Innerhalb des

Applets gibt es nur zwei Dialogelemente, die dazu in der Lage sind eine solche Nachricht auszulösen und dementsprechend eine Reaktion zu provozieren. Diese beiden Elemente sind die Buttons, welche in dem dritten Panel eingebettet sind. Diese Buttons senden eine Action - Nachricht aus, sobald sie betätigt werden. Um die Nachrichten, die von den Buttons ausgesendet werden überhaupt wahrnehmen zu können, müssen diese ebenfalls bei dem entsprechenden Listener registriert sein. Diese Nachrichten werden dann von der einzigen Methode abgefangen, die das Interface des ActionListeners bereitstellt. Dabei handelt es sich um die Methode *public void ActionPerformed*. Dieser wird ein String übergeben, um die jeweilige Nachricht identifizieren zu können. Der String beinhaltet das Label des betätigten Buttons. Alle Aktionen innerhalb des ActionListeners werden also dementsprechend über ein Label gesteuert. Im Folgenden werden nun die einzelnen möglichen Optionen kurz angesprochen und erläutert.

Der linke Button

Zuerst wird sich an dieser Stelle mit dem linken der beiden Buttons befasst. Dieser dient innerhalb des Applets nur selten der eigentlichen Navigation, sondern übernimmt Sonderaufgaben während der Eingabephasen und wird bei benutzerbezogenen Entscheidungen hinzugezogen. Dem zu Folge sind dem linken Button folgende Label zugeteilt:

new: Dieses Label ist dem Button bei der Initialisierung des Applets zugedacht. Es dient der Auswahlmöglichkeit des Benutzers, ob dieser eine neue Enrollmentphase starten möchte oder nur eine Vergleichseingabe durchführen möchte. Wird dabei der Button *new* gewählt, so wird eine neue Enrollmentphase gestartet.

clear: Der linke Button wird mit diesem Label versehen, wenn sich ein Benutzer in einer Eingabephase befindet. Der Button dient dazu alle bisher erfassten Werte innerhalb der konkreten Phase zurückzusetzen, damit der Benutzer die Möglichkeit hat die Phase noch einmal erneut starten zu können. Dies ist insofern sinnvoll, da man bei einer Eingabe unterbrochen oder anderweitig abgelenkt werden kann und dann die Möglichkeit bestehen sollte die einzelne Eingabe erneut starten zu dürfen ohne das gesamte Applet resetten zu müssen.

Compare: Dieses Label leitet einen abschließenden Kommentar ein und wird verwendet, wenn das Applet am Ende seiner Auswertung steht, oder der Benutzer nach der Enrollmentphase gefolgt von der ersten Auswertung kein Interesse mehr hat eine Vergleichseingabe durchzuführen und das Applet an dieser Stelle beenden möchte. Da ein Applet allerdings erst dann beendet wird, wenn der Benutzer den Browser schließt, in dem das Applet arbeitet, wird durch das Label nur ein entsprechender Text ausgegeben und eine weitere Eingabe unmöglich gemacht.

–: Dieses Label ist ohne eine große Funktion und wird an dieser Stelle der Vollständigkeit wegen erwähnt. Es wird eingesetzt um den linken oder auch den rechten Button während einzelner Phasen des Applets außer Kraft zu setzen um eine ungewollte Verfälschung zu verhindern.

Der rechte Button

Der rechte der beiden Buttons dient im Gegensatz zu dem linken Button eher der Navigation innerhalb des Applets. Der Benutzer kann mit ihm zwischen den einzelnen Schritten wechseln. Zusätzlich erfüllt er, wie der linke Button, ebenfalls eine Auswahlfunktion innerhalb des Applets an den entsprechenden Stellen. Folgende Label sind dem rechten Button zugedacht:

known: Dieses Label ist das Pendant zu dem Label *new* des linken Buttons. Wenn dieses Label anstelle des Labels new ausgewählt wird überspringt der Anwender die Enrollmentphase des Applets und veranlasst das Laden von bereits vorhandenen Vergleichsdaten, welche in der externen Datei biodata.text während dem Durchlaufen einer vorherigen Enrollmentphase abgelegt worden sind.

2.type: Dieses Label leitet innerhalb der Enrollmentphase den zweiten Eingabeschritt ein und schließt den ersten Eingabeschritt ab, wobei die TextArea und die TextFields innerhalb der ersten beiden Panels zurückgesetzt werden. Die Indizes der Speicherarrays werden ebenfalls neu gesetzt.

interimEval: Dieses Label leitet innerhalb der Enrollmentphase die abschließenden Auswertungen ein und schließt die zweite Eingabephase ab. Nach dem

Betätigen des Buttons werden dann die Methoden *average( )* und *evaluations( )*, sowie die Methode *Datasave( )* ausgeführt, welche später noch genauer erläutert werden.

CreateMatchingInput: Ist die Auswertung der Enrollmentphase abgeschlossen oder der Benutzer hat sich durch die Auswahl des Buttons *known* dazu entschlossen die Enrollmentphase zu überspringen und stattdessen vorher generierte Daten zu laden, so kann der Benutzer durch das Betätigen des rechten Buttons, wenn dieser das Label *CreateMatchingInput* trägt die zweite Eingabephase starten, bei der der Benutzer eine Vergleichseingabe über die TextArea durchführt.

Compare: Sobald die Vergleichseingabe eingeleitet worden ist wird der rechte Button mit dem Label Compare versehen. Ist die Vergleichseingabe beendet, so wird diese durch das Betätigen des Buttons abgeschlossen. Innerhalb des Schrittes Compare wird dann die abschließende Auswertung durchgeführt und mit den Vergleichswerten aus der ersten Berechnungsphase abgeglichen. Dies geschieht über die Methode compare( ), welche im späteren Verlauf noch eingehender beschrieben wird.

Einen genauen Überblick über die Zusammenhänge zwischen den einzelnen Labeln und der damit verbundenen Navigation gibt die Grafik D.1 wieder.

4.2. Datenauswertung

In diesem Abschnitt wird nun genauer beschrieben, auf welche Weise die gewonnenen Daten aus den einzelnen Eingabephasen verarbeitet und letztendlich auch abgespeichert werden, um diese für eine spätere Wiederverwendung gegenwärtig zu haben.

4.2.1. Auswertung der Referenzdaten - Die Phase „interimEval“

Nachdem nun die Navigation und die damit möglichen Optionen des Applets dargelegt worden sind richtet sich nun die Aufmerksamkeit auf jene Phasen, in denen

das Applet die eingegebenen Daten auswertet und abspeichert. Dabei ist die externe Speicherung mit einigen Problemen verbunden, da Applets von Java als nicht sicher eingestuft sind und ein Zugriff auf ein Hardwarespeichermedium, wie eine Festplatte, von dem in Java integrierten Security Manager nicht zugelassen wird. Bevor allerdings dieses Problem und eine entsprechende Lösung genauer angegangen werden, wird zuerst die interne Datenverarbeitung und die Abspeicherung der gewonnenen Daten erläutert.
Bereits an einigen Stellen wurden Arrays erwähnt, welche mit Hilfe der Implementierung des KeyListeners mit Werten gefüllt werden. Es handelt sich dabei um drei zweidimensionale Arrays(siehe Abbildung 4.1), welche intern dazu verwendet werden die gesammelten Daten des Benutzers aufzunehmen und für die Verarbeitung abzuspeichern. Dabei wird die erste Dimension von dem jeweiligen Arbeitsschritt festgelegt. Dementsprechend gibt es drei verschiedene Dimensionen für die jeweiligen Erfassungsphasen. Die zweite Dimension der Arrays wird dann in den jeweiligen Erfassungsschritten mit den erfassten Werten gefüllt. Dabei kann es theoretisch zu einem Überlauf kommen, da in Java ein Array mit einer festen Angabe für die jeweilige Dimension angelegt werden muss. In diesem Applet ist dieser Wert für die zweite Dimension auf Zweitausend gesetzt. Sollte es also dazu kommen, dass die Laufvariable *secdimensioncount* diesen Wert überschreitet, wird zumindest bei der aktuellen Implementierung, ein entsprechender Overflow - Fehler ausgelöst. Das Applet muss also gegebenenfalls angepasst werden, wenn es einen größeren Datensatz aufnehmen soll.

Quellcodeausschnitt 4.1 ()
```
long [ ][ ] bksave = new long [3][2000];
long [ ][ ] kdsave = new long [3][2000];
int [ ][ ] keycodes = new int [3][2000];
```

Neben diesen drei Arrays wird noch ein weiterer globaler Array mit dem Namen *extension* innerhalb des Applets verwendet. Dieser ist eindimensional und wird zur Speicherung der Ergebnisse aus der ersten Zwischenberechnung verwendet. Auf diese Ergebnisse und wie sie zustande kommen wird nun genauer eingegangen.
Innerhalb des Schrittes interimEval werden die Daten aus der Enrollmentphase verarbeitet. Die Ergebnisse der Auswertung dienen danach als Vergleichswerte für weitere Eingaben und können immer wieder verwendet werden, da diese Ergebnisse in einer externen Datei gespeichert werden, worauf später noch genauer eingegangen werden wird.

Die Methode average()

Innerhalb der Auswertung wird zuerst die Methode *average* aufgerufen. Diese dient der Ermittlung der Mittelwerte der drei Arrays, welche zuvor mit Werten gefüllt worden sind. Da die Berechnung der Werte für die Arrays weitestgehend identisch ist, wird an dieser Stelle nur die Berechnung für den Array *bksave[ ][ ]* genauer beschrieben und nur bei Abweichungen noch genauer auf die Arrays *kdsave[ ][ ]* und *keycodes[ ][ ]* eingegangen.
Die Auswertung jedes einzelnen Array erfolgt innerhalb einer geschachtelten Schleife. Die äußere Schleife wird dabei benötigt um den Index für die erste Dimension festzulegen. Innerhalb der inneren Schleife werden dann die Werte der einzelnen Eingabephasen ausgewertet. Dabei wird allerdings der erste Speicherplatz innerhalb des Arrays nicht beachtet, da dieser nur einen Zeitstempel anstelle eines berechneten Wertes beinhaltet. Für jeden Wert innerhalb des Arrays wird nun überprüft ob dieser ungleich Null ist. Falls er den Wert Null haben sollte wird er nicht in die Berechnung integriert, da dann der entsprechende Speicherort nicht belegt worden ist. Ist die innere Schleife vollständig abgearbeitet, wird aus den Ergebnissen innerhalb der äußeren Schleife der Mittelwert für den jeweiligen Schritt ermittelt und in dem Array *result[ ]* abgelegt. Wenn die äußere Schleife abgearbeitet ist wird dann aus den Ergebnissen, welche in *result[ ]* abgelegt sind, der Durchschnitt beider Schritte errechnet, welcher dann in dem globalen Array *extension[ ]* an der Position Null (siehe 4.2) beziehungsweise für die Ergebnisse von *kdsave[ ][ ]* an der Position Eins und für *keycodes[ ][ ]* an der Position Zwei abgelegt wird.

Quellcodeausschnitt 4.2 ()

```
for (int x=1; x<=2; x++)
{
long mention = 0;
int count = 0;
text.append("Computing average Time between keys for the "+x+". Step!\n ");
text.append("Captured Values:\n");
for(int y=1 ; y<bksave[x-1].length; y++)
{
if (bksave[x-1][y] != 0)
{
tbk.setText(""+bksave[x-1][y]+"\n");
mention += bksave[x-1][y];
count++;
}
}
text.append("Added times for this Step "+mention +"!\n");
text.append(count+" time(s) counted\n");
text.append("Average Time for this Step: "+ mention/count+"\n");
result[x-1] = mention/count;
}
extension[0] = (result[0]+result[1])/2;
text.append("The final average Time between the Keys is "+extension[0]+"\n");
```

Die Methode evaluations()

Die restlichen Werte werden innerhalb der Methode *evaluations()* ermittelt. Anders als in der Methode *average()* werden hier keine Durchschnittswerte ermittelt, sondern es wird das Aufkommen von Abweichungen und die Anwendung der Löschen - Taste genauer bestimmt. Da die Methode entsprechend komplexere Berechnungen durchführt ist der Array *result[ ]*, der auch innerhalb dieser Methode für die lokale Speicherung von Zwischenergebnissen aus den jeweiligen Schritten verwendet wird entsprechend größer, da er bis zu sechs verschiedene Werte aufnehmen muss.
Die ersten beiden Werte, welche innerhalb der Methode berechnet werden, beziehen sich auf die Verwendung der Löschen - Taste, auch Backspace genannt, und deren Verwendung innerhalb einer Eingabe. Um den Aufruf der Taste erkennen zu können wird der Array *keycodes[ ][ ]* nach Vorkommen des virtuellen Codes durchsucht, der die Taste Backspace repräsentiert, die Zahl Acht. Innerhalb der

Auswertung wird nun der Array *keycodes[ ][ ]* nach dem Code durchsucht. Das Programm unterscheidet, ob es sich nur um einen oder zwei aufeinander folgende Vorkommen des Codes handelt. Dabei gibt es allerdings ein bis jetzt noch nicht gelöstes Problem, da das automatische Wiederholen einer Taste, sofern man diese gedrückt hält nur als einmaliges Löschen erkannt wird, obwohl beliebig viele Zeichen gelöscht werden können. Um innerhalb der Verarbeitung keine *ArrayIndexOutOfBoundException* auszulösen muss die letzte Position des Arrays vorher abgefragt werden. Der Quellcodeausschnitt 4.3 zeigt die entsprechenden Zeilen.

Quellcodeausschnitt 4.3 ()

```
for(int x=1;x<=2;x++)
{
int count=0;
int count2 = 0;
text.append("Counting use of Backspace in the "+x+". Step:\n");
if (keycodes[x-1][(keycodes[x-1].length-1)]==8)
{
count ++;
}
```

Dadurch kann es bei diesem besonderen Fall dazu kommen, dass das einzelne Betätigen einer Löschen - Taste erkannt wird, selbst wenn an der vorletzten und letzten Position des Arrays ein entsprechender Code gefunden wird, was allerdings relativ unwahrscheinlich ist, da normalerweise am Ende einer Eingabe nicht mehrere Löschvorgänge vorgenommen werden, es sei denn es wurden mehr Werte erfasst als der Array aufnehmen kann. Nach dem Eintritt in die innere Schleife ist dieses Problem nicht mehr akut. Hier wird zuerst das doppelte Vorkommen einer Löschen - Taste überprüft. Ist dies der Fall, so wird die entsprechende Countervariable erhöht. Sollte diese Überprüfung kein positives Ergebnis zurückgeben, wird äquivalent für das Vorkommen von einer Löschen - Taste vorgegangen, wie es der Quellcodeausschnitt 4.4 zeigt.

Quellcodeausschnitt 4.4 ()

```
for(int y=1; y<(keycodes[x-1].length-1); y++)
{
if (keycodes[x-1][y]==8 && keycodes[x-1][y+1]==8)
{
count2 ++;
y++;
}
else if (keycodes[x-1][y]==8)
{
count++;
}
}
```

Nach dem Durchlauf der inneren Schleife werden noch die Ergebnisse ausgegeben und abgespeichert. Die Ergebnisse aus dem Array *result()* werden dann nach dem Abschluss der Auswertung noch in einer abschließenden Berechnung zusammengefasst, es wird der Durchschnitt aus den beiden Erfassungsschritten ausgegeben und in dem globalen Array *extension[ ]* an den Positionen Drei und Vier abgelegt(siehe Quellcodeausschnitt 4.5).

Quellcodeausschnitt 4.5 ()

```
text.append(count+" Backspaces found in the "+x+". Step!\n");
result[x-1] = count;
text.append(count2+" two Backspaces found in the "+x+". Step!\n");
result[x+1] = count2;
}
extension[3]=(result[0]+result[1])/2;
text.append("Found Backspaces in Average: "+extension[3]+"!\n");
extension[4]=(result[2]+result[3])/2;
text.append("Found two Backspaces in Average "+extension[4]+"!\n");
```

Der nächste Schritt befasst sich genauer mit der Abweichung eines Tastendrucks von einem vorher festgelegten Wert. Dieser Wert ist in der globalen Variable *differ* abgespeichert und aktuell auf fünfundvierzig Millisekunden Abweichung sowohl im positiven, als auch im negativen Rahmen eingestellt. Dabei basiert diese Einstellung auf eigenen Versuchen und wird wahrscheinlich im späteren Verlauf noch angeglichen werden müssen. Durch die Einbindung des bereits in der Methode *average()* errechneten Mittelwertes wird allerdings eine relative Sicherheit bei der Erfassung anzustreben versucht, da sich dadurch der Mittelwert zumindest in ge-

ringem Maß an den jeweiligen Benutzer angleichen lässt. Wie bereits zuvor bei der Berechnung der Mittelwerte wird auch hier auf die Erfassung des Wertes an der Position Null bewusst verzichtet. Innerhalb der geschachtelten Schleife werden nun die vorhandenen Daten aus dem Array *kdsave[ ][ ]* auf die bereits erwähnte Abweichung hin untersucht. Bei einem Fund wird die Laufvariable *count* inkrementiert. Nach der Abarbeitung werden noch entsprechende Abschlussausgaben getätigt und die Ergebnisse in dem globalen Array *extension[ ]* an der Position Fünf abgelegt, wie der Quellcodeausschnitt 4.6 zeigt.

Quellcodeausschnitt 4.6 ()

```
for(int x=1;x<=2;x++)
{
int count = 0;
text.append("Evaluating Differences for the key down time in the ");
text.append(x+". Step.\n");
for(int y=1; y<kdsave[x-1].length; y++)
{
if (kdsave[x-1][y]!=0)
{
if((kdsave[x-1][y]< (extension[1]-differ))||
       (kdsave[x-1][y]> (extension[1]+differ)))
{
count++;
text.append("Difference found at Position "+y+"!\n");
}
}
}
result[x-1] = count;
text.append("The Result for the "+x+". Step is "+count+".\n");
}
extension[5] = (result[0] + result[1])/2;
text.append("Average Result for both steps: " +extension[5]+"\n");
```

Abschließend wird der Array *bksave[ ][ ]* auf Abweichungen untersucht, allerdings werden bei der Untersuchung dieses Arrays nur Abweichungen gegen den oberen Rahmen berücksichtigt, da es ansonsten je nach dem individuellen Verhalten des Benutzers vorkommen kann, dass die untere Grenze den Wert Null unterschreitet, wodurch es während der Laufzeit zu Fehlern kommen kann. Bei der Überprüfung innerhalb dieses Schrittes ist dafür eine geschachtelte Unterteilung implementiert, welche die Abweichung zu dem oberen Rahmen hin in mehrere Gruppen unterteilt.

Dabei wird zuerst überprüft ob die Abweichung von dem Mittelwert, welcher in der Methode *average()* bestimmt worden ist, mehr als 200 Millisekunden beträgt. Ist dies nicht der Fall, so wird überprüft ob die Abweichung des aktuellen Wertes aus dem Array sich zwischen 101 und 200 Millisekunden bewegt. Gibt es auch in dieser Überprüfung kein positives Ergebnis, so wird in einem letzten Schritt eine mögliche Abweichung in einem Zeitfenster zwischen 51 und 100 Millisekunden untersucht. Eine solche Schachtelung macht bei der Überprüfung des Arrays *bksave[ ][ ]* Sinn um zu ermittelten, in wie fern der Anwender während der Eingabe Unterbrechungen macht, um beispielsweise eine weitere Textzeile zu lesen, welche eingegeben werden soll.

Quellcodeausschnitt 4.7 ()

```
for(int y=1; y<bksave[x-1].length; y++)
{
if(bksave[x-1][y]>(extension[0]+200))
{
count++;
text.append("Difference greater than 200 units found at ");
text.append("position "+y+"!\n");
}
else if(bksave[x-1][y]>(extension[0]+100))
{
count2++;
text.append("Difference greater than 100 units found at ");
text.append("position "+y+"!\n");
}
else if(bksave[x-1][y]>(extension[0]+50))
{
count3++;
text.append("Difference greater than 50 units found at ");
text.append("position "+y+"\n");
}
result[x-1] = count;
result[x+1] = count2;
result[x+3] = count3;
}
```

Ist die Überprüfung in der inneren Schleife abgeschlossen, so werden die Ergebnisse des Durchlaufes in dem lokalen Array an den Positionen x - 1, x + 1 und x + 3 abgelegt(siehe Quellcodeausschnitt 4.8). Aus den jeweiligen Ergebnissen des lokalen

Arrays werden abschließend wieder die Durchschnittswerte für die beiden ersten Schritte ermittelt und in dem Array *extension[ ][ ]* an den Positionen Sechs, Sieben und Acht abgelegt.

Quellcodeausschnitt 4.8 ()

```
extension[6] =(result[0] + result[1]) /2;
extension[7] =(result[2] + result[3]) /2;
extension[8] =(result[4] + result[5]) /2;
```

4.2.2. Datenspeicherung und Datenaufruf

Nachdem die Auswertung innerhalb der oben erwähnten Methoden abgeschlossen ist, werden die gewonnenen Ergebnisse, welche in dem Array *extension[ ]* abgelegt worden sind, durch den Aufruf der Methode *Datasave()* in einer externen Datei abgespeichert, um sie später für einen erneuten Datenabgleich eines Benutzers verwenden zu können. Der Speichervorgang gestaltet sich allerdings komplizierter, da Applets bei Java einer anderen Sicherheitspolitik unterliegen als Applikationen.
Java verarbeitet Ein - und Ausgaben mit Hilfe der Klasse *Java.io*. Eine Java - Applikation erhält dabei durch den in Java integrierten Sicherheitsmanager direkt die Erlaubnis auf ein Hardwarespeichermedium zu schreiben oder von diesem zu lesen, da Applikationen von einem Benutzer bewusst gestartet werden und daher davon ausgegangen wird, dass der Benutzer das Schreiben und Lesen von der Festplatte erlaubt hat und dass der Zugriff damit bewusst gewünscht ist. Bei einem Applet ist dies allerdings nicht der Fall. Applets können im Internet durch das Aufrufen einer URL gestartet werden ohne dass der Benutzer dies erlauben muss. Dadurch könnte ein Applet Daten lesen ohne das Einverständnis des Benutzers zu haben. Um diese Sicherheitslücke innerhalb der Java - Umgebung, der Virtual Machine, zu beheben haben die Entwickler von Java Applets keine Erlaubnis zum Schreiben und Lesen erteilt. Da aber die ermittelten Werte mehrmals verwenden zu können ist eine Abspeicherung der Daten zwingend notwendig. Die erforderlichen Rechte der Virtual Machine erhält man nur, indem der Benutzer dem Applet den Zugriff explizit erlaubt. Eine Möglichkeit sich diese Erlaubnis einzuholen ergibt sich durch das Zertifizieren des Applets. Der Benutzer muss das Zertifikat bestätigen um das Applet aufzurufen. Dadurch wird das Applet als benutzergesteuert angesehen und erhält die Erlaubnis zu lesen und zu schreiben.

Die Methode Datasave

Zuerst wird der eigentliche Quellcode innerhalb des Applets genauer betrachtet. Der Code an sich ist recht einfach gehalten. Innerhalb der Methode *Datasave()* wird ein DataOutputStream erstellt, welcher Daten gepuffert in die Datei *biodata.txt* schreibt. Nach der Erstellung werden alle ermittelten Daten innerhalb der Schleife aus dem Array *extension[ ]* in den Puffer geschrieben. Um die Daten extern nicht so einfach lesbar zu machen wurde die Methode *writeFloat()* verwendet, da diese keine konkreten Werte in die Datei schreibt, sondern einen vordefinierten Code verwendet, der von jedem beliebigen Java - DataInputStream gelesen werden kann. Dadurch sind die geschriebenen Daten schwach codiert, was für den Benutzer eine, wenn auch nur geringe, Sicherheit bietet. Ein weiterer Vorteil besteht darin, dass sich die genauen Werte in die Datei schreiben lassen. Andernfalls müssten die Werte erst mit Hilfe einer Wrapper - Klasse in Byte - Werte konvertiert werden, da der OutputStream von Java als Bytestream realisiert ist. Über den Befehl *flush()* werden dann die in dem Puffer gesammelten Werte in die Datei geschrieben. Abschließend wird der OutputStream wieder geschlossen. Wie an dem Quellcode zu sehen ist wird die gesamte Speicherung in einen try - Block geschrieben. Dies ist bei der Arbeit mit der Klasse Java.io zwingend notwendig um den Absturz des Applets zu vermeiden. Nach dem try - Block muss zwingend ein catch - Block folgen, um die eventuell auftretenden Fehler abzufangen. Bei dem Schreiben von Daten müssen zwei mögliche Fehler abgefangen werden. Der erste Fehler wird ausgelöst, wenn die Datei noch nicht vorhanden ist. In diesem Fall erzeugt Java eine neue Datei. Der zweite Fehler wird ausgelöst, wenn die Datei bereits vorhanden ist, aber durch ein Problem das Applet nicht in der Lage ist zu schreiben.

Quellcodeausschnitt 4.9 ()

```
public void Datasave()
{
try
{
DataOutputStream out = new DataOutputStream
   ( new BufferedOutputStream
     ( new FileOutputStream ("biodata.txt") ) );
    for(int x = 0; x<extension.length; x++)
    {
    out.writeFloat(extension[x]);
    text.append("Data exporting\n");
    }
    out.flush();
    out.close();
    }
    catch(FileNotFoundException e)
    {
    text.append("-- File biodata.txt not found --");
    }
    catch(IOException e)
    {
    text.append("-- Unable to write Data! --");
    }
}
```

Die Zertifizierung

Die Methode Datasave ist aber nicht in der Lage Daten abzuspeichern, solange das gesamte Applet nicht die bereits erwähnte Erlaubnis des Benutzers hat. Um diese Erlaubnis zu erhalten bedarf es der ebenfalls bereits erwähnten Zertifizierung des gesamten Applets. Dabei kann aber die reine kompilierte Fassung des Quellcodes, die so genannte class - Datei, nicht direkt mit einem Zertifikat verbunden werden. Daher wird an dieser Stelle kurz beschrieben welche Schritte durchgeführt werden müssen um ein Zertifikat zu erstellen und dieses mit einem Applet zu verbinden. Die genaue Vorgehensweise wird im Anhang genauer erläutert.

1. **Schlüsselpaar erstellen:** Um ein Zertifikat erstellen zu können benötigt man ein Schlüsselpaar bestehend aus einem public key und einem private key. Die

Erstellung eines solchen Schlüsselpaares erfolgt mit dem *keytool* des jdk.

2. **Zertifikat erstellen:** Durch die Eingabe von einigen Daten und dem vorher erstellten Schlüsselpaar wird nun ein Zertifikat erstellt. Die Generierung erfolgt wiederum mit dem *keytool*.

3. **Applet packen:** Um das Zertifikat an das Applet zu binden muss der kompilierte Quellcode in ein jar - Archiv gepackt werden. Dies kann unter anderem mit dem jar - Tool des jdk erfolgen.

4. **Zertifikat einbinden:** Letztendlich gilt es das Zertifikat an das gepackte Applet zu binden um die Zertifizierung abzuschließen. Zu diesem Zweck kann das jarsigner - Tool verwendet werden, welches ebenfalls im Umfang des jdk enthalten ist.

Durch diesen Vorgang ist das Applet nun mit einem Zertifikat verknüpft. Sobald das Applet innerhalb eines Browsers gestartet wird erscheint ein Fenster, welches den Benutzer dazu auffordert das Zertifikat anzunehmen. Sobald dieser das Zertifikat bestätigt hat, wird dem Applet die Erlaubnis erteilt auf persistente Speichermedien zuzugreifen. Dabei ist allerdings zu beachten, dass ein solches Zertifikat nur zeitlich begrenzt gültig ist.[6, 7]

Daten auslesen

Falls ein Benutzer bereits irgendwann zuvor die Enrollmentphase durchlaufen hat, so kann er die dabei abgespeicherten Daten auch einlesen lassen um dann nur die Vergleichseingabe durchzuführen und direkt die Endauswertung einzuleiten. Um dies zu ermöglichen wird anfangs, sofern der Button „known“ ausgewählt worden ist, mit Hilfe eines InputStreamReaders innerhalb der Methode *Dataread()* der Array *extension[ ]* mit den Daten aus der Datei *biodata.txt* gefüllt. Da der Aufbau der Struktur mit der Methode *Datasave()* äquivalent ist, wird an dieser Stelle nicht weiter darauf eingegangen. Es ist wichtig zu beachten, dass das Programm nicht sicherstellen kann, welcher Benutzer zuletzt auf die Datei zugegriffen hat. Ein weiteres Problem besteht darin, dass die Methode bei der aktuellen Fassung auf jeden Fall ausgeführt wird, selbst wenn die Datei *biodata.txt* nicht existiert. In diesem Fall wird ein Fehler ausgelöst, der zwar intern behandelt wird, aber der

zumindest zu diesem Zeitpunkt noch keine für den Anwender sichtbare Ausgabe generiert. Der einzige Anhaltspunkt besteht in der Ausgabe des Arrays, welche generiert wird, bevor die Vergleichseingabe startet, sofern man den Button „known“ ausgewählt hat.

4.2.3. Die letzte Auswertung - Die Phase „Compare“

Nachdem der Benutzer die Vergleichseingabe beendet hat wird diese offiziell mit der Betätigung des Buttons *compare* abgeschlossen und die abschließenden Berechnungen begonnen. Dazu wird die Methode *compare()* aufgerufen. Innerhalb der Methode werden dann die Daten der Vergleichseingabe äquivalent zu den Methoden *average* und *evaluations()* ausgewertet. Zusätzlich zu der Auswertung werden die einzelnen Ergebnisse mit den Werten aus der Enrollmentphase abgeglichen und die Differenz wird dem Benutzer angezeigt. Letztendlich wird dann aus den Ergebnissen der Enrollmentphase und den Ergebnissen der Vergleichsauswertung noch eine prozentuale Übereinstimmung, der Matching Score, errechnet. Die Grafik 4.1 zeigt die Vorgehensweise für jeden einzelnen der neun Werte, welche verwendet werden.

Abbildung 4.1.: Ablaufbeschreibung der Vorgehensweise während der Endauswertung

Wie an der Grafik 4.1 zu sehen ist, wird bei der Berechnung jedes einzelnen Wertes direkt ein Zwischenergebnis ermittelt, welches in die abschließende Berechnung des Matching Scores einbezogen wird. Dabei wird zuerst überprüft, welcher der beiden Werte größer ist. Danach wird der kleinere der beiden Werte durch den größeren geteilt. Ist also beispielsweise der Wert x des Vergleichsdatensatzes größer als der Wert y der aktuellen Auswertung sieht die Rechnung wie folgt aus:

$$\boldsymbol{y / x = z}$$

Die einzelnen Ergebnisse werden dann zusammengefasst und durch die Anzahl der vorhandenen Werte geteilt. Das Ergebnis dieser Berechnung ergibt einen Wert, der zwischen Null und Eins liegt und die Übereinstimmung der beiden Datensätze widerspiegelt. Je größer der Wert ist, umso größer ist die Übereinstimmung zwischen den beiden Datensätzen. Bei der Berechnung der einzelnen z - Werte fließt dabei keine andere Variable ein, die dem jeweiligen Wert noch eine besondere Gewichtung verleiht. Dieser Wert wird abschließend noch mit 100 multipliziert und über das Textfeld als Prozentwert ausgegeben.

5. Anwendungsmöglichkeiten

Nachdem das Applet in dem vorigen Kapitel in seiner vollen Funktionsweise beschrieben worden ist, folgen nun im Verlauf dieses Kapitels einige Überlegungen im Bezug auf mögliche Verwendungszwecke für das Applet. Anbei ist allerdings zu bemerken, dass in dem momentanen Zustand des Applets eine wirklich effektive Anwendung noch nicht sinnvoll ist, da es noch einiger Arbeitsschritte bedarf um einen sinnvollen Einsatz zu ermöglichen. Die im Folgenden vorgestellten Überlegungen sind daher als Ideen für eine zukünftige Anwendung zu sehen.

5.1. Allgemeines

Biometrische Verfahren werden bereits in der heutigen Zeit erfolgreich eingesetzt, vor allem der Fingerprint und Irisscans erlangen zunehmende Popularität. Wie bereits in den Grundlagen erwähnt gehören diese Merkmale allerdings zu der Gruppe der statischen Merkmale, da sich diese nicht durch verhaltensspezifische Muster prägen, sondern durch genotypische Eigenschaften einen Benutzer als Unikat ausweisen. Die Erkennung und Verifizierung eines Benutzers anhand seines Verhaltens bei der Eingabe an einer Tastatur hingegen gehört zu den dynamischen biometrischen Merkmalen. Die unikaten Eigenschaften eines Benutzers stammen nicht zwingend von einer genetischen Veranlagung ab, sondern basieren auf antrainierten Verhaltensmustern im Bezug auf den Umgang mit der jeweiligen Tastatur. Problematisch bei einem solchen Erkennungsverfahren sind die starken Schwankungen denen eine Eingabe unterliegen kann. Im Folgenden werden nun ein paar Faktoren aufgezählt, welche eine Eingabe beeinflussen können:

Hardwaredifferenzen : Die Art und Weise einer Benutzereingabe kann von der Tastatur abhängen, die der Benutzer verwendet. Auf den ersten Blick mag dies

keine Einschränkungen mit sich bringen, allerdings sieht das schon anders aus, wenn ein Benutzer mehrere Arbeitsplätze innerhalb eines Netzwerkes auf seinen Namen anmelden kann und diese Plätze unterschiedliche Tastaturtypen verwenden. So kann zum Beispiel ein Arbeitsplatz ein Festrechner mit einer externen Tastatur sein und ein anderer ein Notebook mit integrierter Tastatur. Die Tastaturen sind mit Sicherheit different. Gleiches gilt auch für die Handhabung des jeweiligen Arbeitsplatzes und die Form der Tastatur. Dadurch kann es zu unterschiedlichen Eingabemustern kommen, welche dann zu einer Abweisung führen können. Des Weiteren sollte das Applet immer unter den gleichen Hardwarebedingungen arbeiten um theoretisch immer die gleichen Zeitwerte ermitteln zu können, denn die Funktion von Java, die für die Ermittlung der Zeitstempel angewendet wird, liefert unterschiedliche Ergebnisse zurück, wenn sie auf unterschiedlich konfigurierten Hardwaresystemen betrieben wird ([4], Kapitel 16.3.5).

Physiologische Einschränkungen : Neben den Hardwareproblemen sind die Probleme, die durch den Benutzer selbst verursacht werden um einiges schwerer zu kompensieren. Dabei unterscheidet man zwischen den physiologischen und den psychischen Veränderungen.
Bei den physiologischen Einschränkungen handelt es sich um jene Probleme bei der Erfassung, die durch den Körper verursacht werden. Wie kann sich zum Beispiel ein Mensch gegenüber einem System sicher verifizieren, wenn er sich am Tag zuvor bei einem Sportunfall die Hand verstaucht hat und nicht wie gewohnt mit beiden Händen schreiben kann. Andere Verletzungen der Hand haben ähnliche Änderungen in dem Muster zur Folge, von eingerissenen Fingernägeln bis hin zum Verlust von Fingern kann jede Verletzung an den Händen potentiell die Erkennung eines Benutzers verhindern. Neben den Händen lassen sich auch noch andere mögliche Einschränkungen auf Grund von Unfällen erdenken wie zum Beispiel gebrochene Arme, ein starres Genick oder Augenprobleme, die die Verifikation des Benutzers erschweren können. Sind solche Verletzungen nicht temporär, sondern von Dauer, so muss man gegebenenfalls nur eine neue Referenzdatei für die Benutzererkennung erstellen, bei temporären Vorfällen allerdings kann man je nach dem nicht regelmäßig neue Referenzdaten erstellen, um die Erkennung immer fehlerfrei ermöglichen zu können.

Psychische Einschränkungen : Während die physiologischen Einschränkungen auf dem direkten Einfluss von anatomischen Anomalien beruhen, geht es bei den

psychischen Eigenschaften um Einflüsse, die auf dem Gemütszustand des jeweiligen Anwenders beruhen. Verhaltensbasierte und antrainierte Eigenschaften wie das Tippverhalten unterliegen im Gegensatz zu statischen Merkmalen mehr oder weniger den emotionalen Schwankungen des jeweiligen Anwenders. Beispielsweise können Frustration oder Müdigkeit die Konzentration des Anwenders beeinträchtigen, was wiederum eine größere Anhäufung von Fehlern mit sich bringt, welche dann letztendlich das Tastaturbild abfälschen. Andere mögliche Einflüsse können auf Krankheiten basieren, welche den Organismus des Menschen schwächen, ihn daher von seiner Arbeit ablenken und dadurch seine kognitiven Fähigkeiten beeinträchtigen, oder durch den Konsum von beeinflussenden Mitteln wie Alkohol und Medikamenten hervorgerufen werden.

5.2. Verifikation eines Benutzers

Ein großer Vorteil der Erkennung über die Tastatur ist der geringe Aufwand bei der Installation. So ziemlich jeder Rechner verfügt über eine Tastatur. Daher bedarf es bei der Verwendung des Applets keiner zusätzlichen Hardware. Die Probleme und daraus resultierenden Nachteile der Verwendung einer Benutzererkennung sind oben bereits beschrieben. Für einen täglichen Einsatz ist das Applet allerdings weniger geeignet, da ein Benutzer bei einer sich kontinuierlich wiederholenden Eingabe zu einer gewissen Routine neigt, die das Eingabemuster abfälschen kann. Bei dem Applet werden stets Abgleiche basierend auf dem Datensatz durchgeführt, der aus der Enrollmentphase vorliegt. Andere Systeme, wie zum Beispiel Psylock, kompensieren diesen Faktor durch einen permanent laufenden Lerneffekt, welcher die Veränderungen wahrnimmt und in den sich ebenfalls dadurch ändernden Datensatz des Templates aufnimmt. Da dieser Faktor bei dem Applet zusätzlich zu den bereits bestehenden Problemen bei der Verwendung der Tastenerkennung hinzu käme werden daher im Folgenden Möglichkeiten beschrieben, für die das Applet in seiner Verwendung eher geeignet ist, da es sich nicht um alltägliche Anwendungen handelt, sondern um sporadisch anfallende Aufgaben, bei denen die Erkennung und Verifizierung eines Benutzers sinnvoll und hilfreich sein kann.

5.2.1. Passworterneuerung

Eine Möglichkeit das Applet zu verwenden wäre die Erkennung eines Benutzers, falls dieser sein Passwort bei einer Internetanwendung nicht mehr zugegen hat und deshalb für einen bereits bestehenden Account ein neues Passwort anlegen muss. Dabei kann das Applet zur Verifizierung des Benutzers zwischengeschaltet werden. So kann der Benutzer nach der Eingabe seines Benutzernamens sein neues Passwort nach einer erfolgreichen Bestätigung des Applets eingeben anstatt sich das Passwort zusenden zu lassen oder einen neuen Account anlegen zu müssen. Dies kann vor allem dann hilfreich sein wenn der Benutzer zum Beispiel den Account für einen entsprechenden Onlineservice schon länger besitzt und in der Zwischenzeit eine andere E - Mail Adresse verwendet. Sollte er vergessen haben diese Adresse bei dem Account auf dem neuesten Stand zu halten, so kann es passieren, dass er bei dem Verlust seines bisherigen Passwortes seinen Account nicht wieder verwenden kann, da in den meisten Fällen ein Internetlink oder aber das Passwort an die im Account angegebene E - Mail Adresse versendet wird. Implementiert man allerdings die Möglichkeit der Passworterneuerung unter der Verwendung der Tastaturerkennung, so kann der jeweilige Benutzer seinen Account verwenden, ohne ihn permanent pflegen zu müssen, wodurch das unnötige Anlegen eines zweiten Accounts und dadurch Datenredundanz vermieden wird.

5.2.2. Dokumentsignierung

Eine weitere Möglichkeit wäre die „Signierung“ eines Dokumentes, also die Sicherstellung, dass ein Dokument, wie zum Beispiel eine Studienarbeit oder behördliche Unterlagen auch wirklich von der Person stammen, die vorgibt dieses Dokument verfasst zu haben. Das Applet in einer abgewandelten Form kann bei einer solchen Dokumentverifikation angewendet werden, indem man es im Hintergrund laufen lässt, während das Dokument erstellt wird. Dabei werden sporadisch einzelne Sätze überprüft und mit einem entsprechenden Referenzmuster abgeglichen. Dadurch kann überprüft werden ob eventuell nicht befugte Personen an einem Dokument, an einer Internetseite oder ähnlichem partizipiert haben. Eine solche Überwachung kann auch bei der Arbeit in einer größeren Gruppe von Interesse sein um zu überwachen, ob ein Mitglied auch nur die Änderungen an einem Gruppendokument vorgenommen hat, zu der das jeweilige Mitglied aufgefordert und berechtigt gewe-

sen ist. Es wird zwar keine vollständige Sicherheit erreicht werden können, aber man kann mit einer größeren Wahrscheinlichkeit den Missbrauch von Informationen und versuchte betrügerische Akte nachweisen.

5.2.3. Arbeitsplatzsicherung

Eine dritte und an dieser Stelle letzte vorgestellte Anwendungsmöglichkeit für das Applet wäre eine Sicherung des eigenen Arbeitsplatzes während der Abwesenheit. Arbeitet man zum Beispiel in einem großen Büroraum zusammen mit mehreren Personen und muss für kurze Zeit seinen Arbeitsplatz verlassen, so wird nicht jeder zuerst seinen Arbeitsplatz ordnungsgemäß abmelden. Um zu verhindern, dass ein Unbefugter in Abwesenheit des Benutzers sich an dessen Arbeitsplatz, Dokumenten oder Einstellungen zu schaffen macht könnte man das Applet im Hintergrund laufen lassen und dazu verwenden die erste Eingabe nach einem gewissen, vorher festgelegten Zeitraum zu überprüfen. Entspricht diese nicht dem typischen Muster des Anwenders, so kann der Rechner zum Beispiel gesperrt werden, so dass der Benutzer sich dann erneut mit seinem Namen und Passwort anmelden muss. Dadurch sind die Daten des Benutzers gesichert und man kann in einer erweiterten Fassung sogar versuchen zu ermitteln, welcher Benutzer versucht hat auf das ihm fremde System zuzugreifen. Dazu bedarf es allerdings einer Datenbank um die Daten mit allen vorhandenen Referenzmustern abzugleichen. Eine solche Identifikation ist allerdings nicht so einfach wie die Verifikation und es bedarf noch einiger Erweiterungen an dem Applet um eine solche Anwendung zu ermöglichen.

Weitere Einsatzmöglichkeiten für die Anwendung des Applets sind sicherlich ebenfalls denkbar. Allerdings soll dieses Kapitel nur eine kleine Übersicht für mögliche Einsatzvariationen der Tastenerkennung geben und die Hindernisse kurz aufzeigen, die bei der Verwendung des Applets bis dahin noch behoben werden müssen. Im Internet gibt es bereits funktionstüchtige Systeme, die die Verarbeitung von biometrischen Werten, gewonnen aus dem Tippverhalten, verwenden um eine Benutzererkennung zu ermöglichen, wie das bereits erwähnte Projekt Psylock[9].

6. Erweiterungsmöglichkeiten und ausstehende Arbeitsschritte

Im vorigen Kapitel wurden ein paar mögliche Anwendungen vorgestellt. Bis zu dem Zeitpunkt, an dem eine Anwendung des Applets realistisch erscheint gilt es noch einige nötige Erweiterungen vorzunehmen. In diesem Abschnitt werden daher einige mögliche Änderungen und Erweiterungen vorgestellt um mehr Sicherheit, Flexibilität und Effektivität zu erreichen.

6.1. Datenspeicherung

Ein Problem bei der Datensammlung ergibt sich durch die statische Implementierung der einzelnen Arrays innerhalb des Applets. Zu dem momentanen Zeitpunkt ist der Aufbau vollkommen ausreichend, da es eine Eingabe von mehr als zweitausend Zeichen benötigt um den Überlauf eines Arrays zu provozieren. Allerdings kann es bei zukünftigen Anwendungen dazu kommen, dass es je nachdem von Nöten ist mehr Daten einzulesen, wie zum Beispiel bei der Überwachung während der Erstellung eines größeren Dokumentes. Ein anderes, allerdings nicht so schwerwiegendes Problem liegt in dem momentan zu hohen Speicherbedarf, den das Applet auf Grund der statisch angelegten Arrays benötigt, denn egal wie umfangreich die endgültige Eingabe innerhalb eines Schrittes ist, es wird immer der gleiche Speicherplatz reserviert.
Eine zweite und schwerwiegendere Aufgabe ergibt sich im Hinblick auf die externe Datenspeicherung. Diese funktioniert zwar, allerdings ist die Art und Weise für eine spätere Anwendung nicht effektiv, da immer nur ein aktueller Nutzer einen Vergleichsdatensatz zur Verfügung hat um mit dem Applet zu interagieren. Es fehlt die Möglichkeit mehrere verschiedene Templates abzuspeichern um eine effektive Verwendbarkeit zu ermöglichen. Dazu bedarf es der Implementierung einer Daten-

bank und einem Benutzerlogin, damit ein Datenschlüssel vorhanden ist um die Daten aus der Datenbank mit einem Benutzer in Verbindung zu setzen. Bei der Implementierung der Datenbank sollte darauf geachtet werden, dass diese genau wie das Applet auch eine Unabhängigkeit gegenüber der Benutzerplattform bietet um eine möglichst variable Funktionalität erreichen zu können. Diese Implementierung wird ein recht großer Schritt für das praxisorientierte Arbeiten mit dem Applet sein, da man von einer „Ein - Benutzer - Verifikation“ zu einer möglichen Verifikation von mehreren Benutzern wechseln kann.

6.2. Verbesserung und Erweiterung der Algorithmen

Neben der Datenspeicherung gibt es auch an der Auswertung der einzelnen internen Datensätze noch einge offenliegende Kapazitäten, die noch nicht implementiert worden sind, oder deren Implementierung noch erweiterbar ist.
Zuerst wäre da eine Erweiterung der bereits bestehenden Auswertung der Zeitwerte anzusprechen, die in den beiden Arrays *kdsave[ ][ ]* und *bksave[ ][ ]* abgelegt worden sind. Die einzelnen Zeiten, welche dort als Differenz angegeben worden sind basieren auf Versuchen, die anhand von Eigenversuchen ermittelt wurden. Allerdings kann man mit Sicherheit nicht davon ausgehen, dass diese Werte dadurch bereits eine gewisse Konsistenz mit sich bringen. Eine optimalere Lösung wäre es zu versuchen die einzelnen Abweichungsvariablen mit Hilfe von einem Algorithmus auf den einzelnen Benutzer zumindest bei einer Verifikation persönlich zuzuschneiden anhand der ermittelten Werte aus den ersten beiden Eingabephasen. Inwiefern ein solcher Algorithmus im Rahmen des Möglichen ist, ist zumindest an dieser Stelle von meiner Position aus noch nicht überprüft worden. Eine wahrscheinlich einfacher zu realisierende Lösung wäre eine feinere Schachtelung der einzelnen Abweichungswerte um genauere und damit auch aussagekräftigere Werte zu erhalten, aus denen dann die auffälligsten Werte extrahiert werden können, welche dann bei der eigentlichen Verifikation eingebunden werden. Eine weitere interessante Aufgabe wäre der Versuch die Zeitarrays in einen Bezug zu dem Zeichenarray zu bringen. Zum Beispiel könnte man dadurch ermitteln ob ein Benutzer immer dann besonders viel Zeit zwischen zwei Zeichen benötigt, wenn er ein bestimmtes Sonderzeichen oder einen eher selten verwendeten Buchstaben tippen muss. Sicher gibt es noch andere Möglichkeiten mit den Zeitarrays zu arbeiten, jedoch soll dieser Abschnitt dazu nur eine Anregung zur Vertiefung und genaueren Ausarbeitung der Arrays bieten,

um eine sinnvollere Nutzung für den Benutzer zu ermöglichen und das Potential der Arrays noch besser auszuschöpfen.
Neben den Zeitarrays kann sich auch noch eingehender mit dem Zeichenarray befasst werden. Momentan wird dieser nur verwendet um den durchschnittlichen Zeichenverbrauch und die Verwendung der Löschen - Taste zu ermitteln. Bei der Auswertung der Verwendung der Löschen - Taste kann man entsprechend noch genauer darauf eingehen. Im Moment werden ausschließlich einzelne und doppelt vorkommende Aufkommen der Löschen - Taste ermittelt. Eine Erweiterung auf eine entsprechende Häufung von Tasten würde wiederum zu genaueren Ergebnissen führen. Des Weiteren muss man bei der Methode noch beachten dass es im momentanen Zustand zu Sonderfällen kommen kann die zu leichten Abweichungen des Wertes führen.
Eine recht große Aufgabe, die noch mit dem Zeichenarray und eventuell noch einem zweiten Zeichenarray zu realisieren wäre, befasst sich mit einer kontextsensitiven Erkennung von Zeichenanomalien, wie zum Beispiel das Buchstabendrehen oder ungewolltes verwenden von Groß - oder Kleinbuchstaben. Um diese Aufgabe bewältigen zu können muss für den Zeichenarray eine Referenz zur Verfügung gestellt werden, mit der die einzelnen Zeichen dann verglichen werden können. Es wäre auch interessant die Reaktion des Benutzers zu erfassen, wenn er während der Eingabe einen Fehler begeht und dann darauf reagiert. Bei der Ermittlung der Daten stellt die Erfassung der Großbuchstaben zumindest im Moment noch ein Problem dar, da der Array *keycodes[ ][ ]* nur den entsprechenden Code für ein Zeichen abspeichert. Der Code allerdings unterscheidet nicht zwischen Groß - und Kleinbuchstaben, da dieser für beide Zeichen gleich ist. Eventuell bedarf es deshalb noch der Implementierung eines weiteren Arrays, der anstelle der Codes die direkt verwendeten Zeichen abspeichert, oder einer Abfrage, ob der virtuelle Code für die Shift - Taste vor der Verwendung einer Buchstabentaste verwendet worden ist. Wie bei den Vorschlägen zu den Zeitarrays ist dies nur eine kurze und mit Sicherheit noch erweiterbare Liste an möglichen Verbesserungen für das Applet um effektivere und genauere Ergebnisse zu erhalten.
Ein anderer Aspekt, welcher in der aktuellen Fassung des Applets nicht überprüft worden ist befasst sich mit dem Einfluss von Umgebungsvariablen. Zum Beispiel die Verwendung des Mauszeigers, um im dem eingegebenen Text hin und her zu springen oder der Gebrauch des „Clear“ - Buttons. Um dies zu ermöglichen werden wahrscheinlich noch weitere Listener in das Applet eingebunden und entsprechende neue Speicherräume geschaffen werden müssen.
Alles in allem sollte versucht werden eine Erweiterung der bestehenden Algorithmen zu erreichen und dabei sollte eine gewisse Interaktion zwischen Zeichen und

Zeiten angestrebt werden.

6.3. Konsistenzermittlungen

Ein dritter und wichtiger Punkt beschäftigt sich mit der Ermittlung der Konsistenz der einzelnen gewonnenen Daten. Um dies zu ermöglichen fehlt es noch an Testläufen unter verschiedenen Bedingungen, welche die einzelnen Daten auf ihre Aussagekraft hin überprüfen. Mit Hilfe einer solchen Prüfung kann man dementsprechend die FAR und die FRR für die Algorithmen ermitteln und dann Spielräume innerhalb des Applets definieren, unter denen ein Benutzer als akzeptiert angesehen wird.
Ein weiterer interessanter Aspekt bei solchen Testläufen wäre eine Überprüfung, inwiefern zum Beispiel die Herkunft eines Benutzers sich über spezifische Schreibmuster nachweisen ließe oder unterschiedliche Tastaturen das Verhalten des Benutzers beeinflussen. Eine solche Überprüfung wäre sinnvoll, wenn ein Benutzer mehrere Arbeitsplätze zur Verfügung hat und diese nicht identisch ausgestattet sind.

6.4. Praxiseinbindung

Letztendlich fehlt es noch an einer realistischen Umgebung in der das Applet eine Anwendung finden kann. Dieser Punkt wird wahrscheinlich erst in einem späteren Stadium angegangen werden können, da das Applet bis zu diesem Zeitpunkt noch der oben beschriebenen Erweiterungen bedarf. Einige mögliche Anwendungsumgebungen wurden ja bereits in dem vorigen Kapitel angerissen um ein repräsentatives Modell zu erhalten, unter dem das Applet eingesetzt werden kann.

7. Fazit

In den vergangenen Kapiteln wurde ausgiebig auf das Erstellen und Arbeiten mit dem Applet eingegangen und die Entstehung sowie der Aufbau der einzelnen Komponenten wurden beschrieben. Innerhalb dieses Kapitels werden nun noch einige abschließende Meinungen und Erfahrungen von meiner Seite aus im Bezug auf das Erstellen dieser Studienarbeit dargelegt.
Ein großes Problem, was sich auch in dem beigelegten Literaturverzeichnis widerspiegelt, ist das relativ aufwändige und meist ergebnislose Suchen von Informationen zu dem eigentlichen Thema der Tastaturerkennung. Es ist zwar nicht weiter schwierig ausgiebige Informationen über Biometrie im Allgemeinen und die recht populären Vertreter biometrischer Verfahren wie Irisscans oder Fingerprints zu erhalten, allerdings scheint die Verifizierung einer Person über die Tastatur eine eher untergeordnete Rolle zu spielen, da das Verfahren aufgrund seines dynamischen Ursprungs nicht sonderlich leicht zu erarbeiten ist. Dementsprechend ist es schwieriger an Informationen zum Stand von Forschung und Entwicklung der wenigen bereits existierenden Systeme oder an Ergebnisse von bereits erfolgreichen Tests zu gelangen, welche die Konsistenz der gewonnen Daten belegen könnte. Die Werte innerhalb des Applets, welche von mir für die Berechnungen der Abweichung von den Mittelwerten angegeben worden sind basieren daher auch nur auf eigenen Versuchen. Eine weitere Frage, welche ich zu dem Zeitpunkt der Entwicklung nicht klären konnte ist die Wandelbarkeit, der das Schriftverhalten eines Menschen im Lauf der Zeit unterliegt. Bei anderen biometrischen Verfahren ist man zum Beispiel in der Lage festzustellen von welcher Dauer die erstellten biometrischen Templates für einen Benutzer gültig sind, bevor diese erneuert werden müssen.
Ein weiteres Problem während dem Erstellungsvorgang des Programms war die oft langwierige Suche nach kleinen Fehlern innerhalb des Quellcodes und die Kontrolle der einzelnen Rechenschritte auf ihre Korrektheit hin. Der Quelltext sieht zwar nach außen hin nicht sehr komplex aus, allerdings steckt der Teufel im Detail. Vor allem bei der Erstellung und dem Testen der Ausgabe- und Einlesemethoden war ein recht hoher Zeitaufwand von Nöten, da nach jeder Änderung am Quellcode das

Applet erneut in ein Archiv gepackt werden und anschließend zertifiziert werden musste.
Diese Arbeit bildet eine Grundlage für weitere Arbeiten, zu denen ich bereits in dem vorigen Abschnitt einige Anregungen gegeben habe, da das Applet in seiner momentanen Fassung eine Art Rohling darstellt. Es bietet eine im momentanen Zustand funktionstüchtige Oberfläche an, mit der ein Benutzer interagieren kann, sowie die Möglichkeit Daten abzuspeichern und wieder abzurufen. Die eigentliche Verifizierung ist meiner Meinung nach mit der Verwendung der momentan gewonnenen Daten eher schwach, da nur neun Erkennungsdaten aus der Eingabe extrahiert werden. Neben der geringen Anzahl von Werten ist es auch noch nötig die Berechnung des Matching Scores zu überarbeiten und entsprechende Gewichtungen einzubauen, damit die einzelnen gewonnenen Ergebnisse ihrer Gewichtung nach angemessen eingebunden werden können. Dabei ist allerdings zu beachten,dass die Einbindung entsprechende Untersuchungen voraussetzt.
Die Verwendung von biometrischen Daten findet in der heutigen Zeit immer mehr Anklang in den verschiedensten Gebieten. Die Erkennung eines Benutzers anhand seines individuellen Verhaltens im Umgang mit einer Tastatur hat mit Sicherheit eine Zukunft aufgrund des geringen Hardwareaufwands, der betrieben werden muss und weil entsprechende Software nicht auf komplexe Bildverarbeitungsprozesse zurückgreifen muss, was auch eine Operabilität auf Systemen mit relativ geringen Leistungsparametern zulässt. Es gilt zwar noch einige Hürden zu nehmen, aber das Applet kann dann letztendlich beispielsweise im universitätsinternen Rahmen ähnlich dem Projekt psylock der Universität Regensburg[9], oder in der freien Wirtschaft einen sinnvollen Zweck erfüllen.

Literaturverzeichnis

[1] **Neuer Reisepass.de**
http://www.neuer-reisepass.de/
Stand 15 Januar 2007

[2] **Keystroke Biometric Recognition Studies on Long-Text Input over the Internet**
von Mary Villani, Mary Curtin, Giang Ngo, Justin Simone, Huguens St. Fort, Sung-Hyuk Cha, and Charles Tappert
erstellt an der CSIS, Pace University, Pleasantville, New York, 10570 USA

[3] **Bromba.com**
http://www.bromba.com/faq/biofaqd.htm
Stand 22 Januar 2007

[4] **Handbuch der Java-Programmierung**
Handbuch der Java-Programmierung 3.Auflage
HTML-Ausgabe 3.0 · ©1998-2002 Guido Krüger
Addison-Wesley, 2002, ISBN 3-8273-1949-8
Homepage: www.javabuch.de

[5] **java.com**
http://www.java.com/de/
Stand 15 Januar 2007

[6] **Signed Applet Tutorial**
http://www-personal.umich.edu/∼lsiden/tutorials/signed-applet/signed-applet.html
Stand 15 Januar 2007
Autor: Larry Siden

[7] **Java Dokumentation**
Java Dokumentation des jdk 1.5.0_09

[8] **Java ist auch eine Insel**
HTML - Buch http://www.galileocomputing.de/openbook/javainsel5/index.htm
Autor: Christian Ullenboom
Version 5

[9] **psylock**
http://www.psylock.de/
Stand 15 Januar 2007

A. Genaue Beschreibung der GUI

Um bei der Erstellung der Benutzeroberfläche eine gewisse Ordnung zu bewahren und die Anordnung im späteren Verlauf besser strukturieren zu können, wurden die einzelnen Abschnitte in Panels zusammengefasst. Die Java - Klasse Panel in der Hierarchie der Fensterklassen wird aus der Klasse Container abgeleitet(siehe Abb. 2.2). Aus Panel wiederum leitet sich die Klasse Applet ab. Ein Panel kann dazu verwendet werden um eine Sammlung von verschiedenen Dialogelementen zu einem größeren Dialogelement zusammenzufassen. Im Verlauf der Erstellung der Benutzeroberfläche wurde diese Eigenschaft genutzt, um die verschiedenen Elemente zusammenzufassen und dann diese einzelnen Panels miteinander zu kombinieren, worauf später genauer eingegangen wird. Im Folgenden werden nun die drei Panels vorgestellt, aus denen sich das Applet zusammensetzt und deren Aufbau sowie Implementierung dargestellt.

A.1. Das erste Panel

Das erste Panel beinhaltet nur ein Element, die TextArea, die dazu dient die einzelnen Texte, welche der Benutzer eingibt, grafisch wiederzugeben und die Weiterleitung der einzelnen Werte mit den Hintergrundprozessen zu ermöglichen. In Java können TextAreas dazu verwendet werden einen mehrzeiligen Text wiederzugeben. Die Klasse TextArea erzeugt dabei ein mehrzeiliges Textfeld. Man kann bei dem Aufruf des entsprechenden Konstruktors optional angeben, welche Maße das Textfeld haben soll, sowie die Möglichkeit für verschiedene Optionen zum Scrollverhalten des Feldes. Das in dem Panel verwendete Textfeld hat folgende Instantiierung:

Quellcodeausschnitt A.1 ()

```
public TextArea text = new TextArea("",10,60,1);}
```

Die beiden Anführungszeichen an der ersten Position stellen sicher, dass das Textfeld leer ist. Optional kann an dieser Stelle ein String eingesetzt werden, welcher dann bei der Erzeugung des Textfeldes mit ausgegeben wird. Die nächsten beiden Werte geben einen Richtwert für die Fenstergröße an, der erste der beiden Werte steht für die erlaubte Zeilenlänge, der zweite gibt die Menge der Spalten an, die sichtbar sind. Diese Werte sind allerdings für das Applet nicht relevant, da die Werte durch das verwendete Layout, auf das weiter unten genauer eingegangen wird, überschrieben werden. Es ist allerdings zwingend notwendig diese mit anzugeben, da Java keinen Konstruktor bereitstellt, der es erlaubt nur einen Wert für die Scrollbars zu setzen. Der letzte Wert steht, wie schon erwähnt, für die Anordnung der Scrollbars, welche innerhalb des Textfeldes zulässig sind. Der Wert 1 steht hierbei für die Konstante „SCROLLBARS_VERTICAL_ONLY“. Dieser Wert erlaubt also nur den Balken an der rechten Seite des Feldes. Der Balken an der unteren Seite des Feldes ist nicht aktiv. Daher kann innerhalb des Feldes ein Textüberlauf über den sichtbaren Bereich der Spalte hinaus nicht erfolgen und ein automatischer Zeilenumbruch tritt ein. Optional können an dieser Stelle noch folgende Werte angegeben werden.

1. **SCROLLBARS_BOTH**: Erlaubt die Verwendung des Textüberlaufs sowohl vertikal als auch horizontal

2. **SCROLLBARS_HORIZONTAL_ONLY**: Erlaubt nur den horizontalen Textüberlauf

3. **SCROLLBARS_NONE**: Erlaubt keinen Textüberlauf

Um innerhalb eines Panels die einzelnen Elemente entsprechend anordnen zu können werden in Java verschiedene Layoutmanager verwendet. Im ersten Panel wird ein GridLayout angewandt. GridLayouts richten die einzelnen Elemente anhand eines vorher festgelegten Gitters aus. Beim Aufruf des GridLayout - Managers werden entsprechend Werte für Zeilen und Spalten angegeben um die Größe des Gitters fest zu legen. Danach werden dann die einzelnen Elemente zu dem Grid hinzugefügt, dabei werden dann die einzelnen Elemente nach und nach in das Gitter eingefügt. In dem ersten Panel wird dem Grid nur ein einzelnes Element mit der

Methode add übergeben, welches die gesamte Größe des Panels ausfüllen soll, daher wird das GridLayout mit den Werten (1,1) aufgerufen (siehe Quellcodeausschnitt A.2).

Quellcodeausschnitt A.2 ()

```
p1.setLayout(new GridLayout(1,1));
p1.add(text);
...
text.addKeyListener(this);
```

Nach der Kompilierung ergibt sich dann folgender Ausschnitt der Benutzeroberfläche:

Abbildung A.1.: Darstellung des ersten Panels

Zusätzlich zu der eigentlichen Implementierung des Textfeldes wird an dieser Stelle noch ein EventListener mit dem Textfeld verknüpft (siehe Quellcodeausschnitt A.2, Zeile 5), auf den noch im späteren Verlauf bei der Erfassung und Sammlung von Daten weiter eingegangen wird.

A.2. Das zweite Panel

Das zweite Panel beinhaltet vier Objekte der Klasse TextField. Im Gegensatz zu einer TextArea gibt es in einem TextField nicht die Möglichkeit einen mehrzeiligen Text auszugeben. Ein TextField ist also eine einzeilige TextArea. Ähnlich wie bei einer TextArea kann man optional die Größe des Feldes angeben und bei Bedarf einen String übergeben. Innerhalb des Panels werden folgende vier Objekte angeordnet:

Quellcodeausschnitt A.3 ()
```
tk.setEditable(false);
cc.setEditable(false);
kdt.setEditable(false);
tbk.setEditable(false);
```

Das erste Feld gibt an, wie viele Zeichen bereits eingegeben worden sind. Bei jeder Art von Eingabe, sei es nun ein Buchstabe, ein Sonderzeichen oder eine Funktionstaste, wird ein Inkrement ausgelöst, das innerhalb dieses Feldes angezeigt wird. Dieses Feld wird bei dem Aufruf mit dem Wert „0“ initialisiert, da noch kein einziges Zeichen bis zu diesem Zeitpunkt von einem Anwender eingegeben worden ist.
Das zweite Feld zeigt das letzte Zeichen an, welches von dem Anwender eingegeben worden ist. Falls das letzte Zeichen eine Funktionstaste gewesen ist, erscheint ein Symbol als Ausgabe. Auf die genauere Behandlung der einzelnen Zeichen und wie diese erfasst werden wird später genauer eingegangen werden.
Die letzten beiden Felder in dem Panel zeigen Zeitwerte an. Das erste Feld zeigt die Zeit an, die der letzte Tastendruck gedauert hat, das zweite Feld die letzte ermittelte Zeitspanne zwischen dem Drücken zweier Tasten. Beide Zeiten werden in Millisekunden angegeben und errechnen sich aus Zeitpunkten.
In diesem Panel wird, genau wie im ersten Panel, ein GridLayout als Layout - Manager verwendet. Da dieses Mal vier Elemente und zusätzlich noch die entsprechenden Label der Elemente innerhalb des Panels untergebracht werden müssen wird der Grid in diesem Fall mit den Werten (4,2) initialisiert. Innerhalb des Panels steht also ein Gitter mit vier Zeilen und zwei Spalten zur Verfügung, welches nun entsprechend gefüllt werden muss. Bei dem Füllen muss beachtet werden, dass der Layout - Manager nach und nach die einzelnen Spalten füllt. Es wird also zuerst das Gitterfeld (1,1), dann das Gitterfeld (1,2) und so weiter bis hin zu dem Element (4,2) eingefügt. Innerhalb der ersten Spalte stehen jeweils die einzelnen

Label, welche dem Anwender anzeigen, wozu die Ausgaben in dem jeweiligen Feld dienen. In der zweiten Spalte sind die zugehörigen Felder angelegt. Da ein GridLayout verwendet wird nehmen die Felder dementsprechend den gesamten Bereich des Gitterfeldes ein. Die Anordnung und der Aufbau des zweiten Panels ist mit Hilfe des in dem Quellcodeausschnitt A.4 realisiert worden.

Quellcodeausschnitt A.4 ()

```
p2.setLayout(new GridLayout(4,2));
p2.add(new Label("Total Keys"));
p2.add(tk);
p2.add(new Label("Current char"));
p2.add(cc);
p2.add(new Label("Key down time"));
p2.add(kdt);
p2.add(new Label("Time between Keys"));
p2.add(tbk);
```

Zusätzlich zu den oben gezeigten Angaben sind die Textfelder noch zusätzlich so eingestellt, dass man die einzelnen Felder nicht extern editieren kann. Dadurch ist es nicht möglich die Angaben innerhalb der einzelnen Felder abzuwandeln, was auch nicht sinnvoll ist, da die einzelnen Felder nur zur Ausgabe dienen und dementsprechend eine Editierung höchstens zu einer Verwirrung für den Anwender führen kann. Diese Einstellung wird mit der Methode *setEditable(boolean)* vorgenommen. Ist setEditable auf *false* gesetzt, so ist das editieren in dem entsprechenden Feld untersagt, andernfalls ist es erlaubt. *True* ist der default - Wert für alle Felder. Im Applet sieht dies folgendermaßen aus:

Quellcodeausschnitt A.5 ()

```
tk.setEditable(false);
cc.setEditable(false);
kdt.setEditable(false);
tbk.setEditable(false);
```

Wenn man das Panel mit den entsprechenden Einstellungen kompiliert und ausführen lässt ergibt sich folgendes Bild:

Abbildung A.2.: Bildausschnitt des zweiten Panels

A.3. Das dritte Panel

Das dritte Panel beinhaltet die beiden Buttons, über die die einzelnen Phasen des Applets gesteuert werden, beziehungsweise mit deren Hilfe man seine Eingabe resetten kann, um noch einmal erneut Werte eingeben zu können. Der linke der beiden Buttons dient dabei vorwiegend dem Resetten und an wenigen Stellen, an denen der Benutzer eine Entscheidung treffen muss, der Navigation, sowie dem endgültigen Beenden des Applets, was aber nur als symbolische Eingabe gedacht ist, da sich Applets zerstören, wenn der entsprechende Browser geschlossen wird. Der rechte Button dient der Navigation innerhalb der Eingabe und wechselt entsprechend der einzelnen Phasen innerhalb des Applets das Label, über das die Buttons innerhalb der Verarbeitung abgefangen werden. Was in den einzelnen Phasen geschieht und wie der Ablauf des Applets von statten geht wird noch genauer behandelt werden. Die Implementierung der Buttons geschieht mit den folgenden Textzeilen:

Quellcodeausschnitt A.6 ()

```
Button leftButton = new Button("new");
Button rightButton = new Button("known");
```

Die Buttons werden innerhalb des Applets mit einem FlowLayout angeordnet, anders als bei einem GridLayout positioniert ein FlowLayout die einzelnen Elemente von links nach rechts innerhalb des Panels an, ohne diese dabei an einem Gitter auszurichten, allerdings ist es möglich die Anordnung in geschwächter Form zu beeinflussen. Man kann optional mit einer der Konstanten *FlowLayout.CENTER*, *FlowLayout.LEFT* oder *FlowLayout.RIGHT* angeben ob die Elemente zentriert,

linksbündig oder rechtsbündig angeordnet werden sollen. Innerhalb dieses Panels werden die Elemente rechtsbündig angeordnet. Des Weiteren kann man optional Angaben zu dem Abstand zwischen zwei Dialogelementen machen, indem man zusätzlich zu den Konstanten für die Anordnung noch Werte für Abstände zwischen den einzelnen Elementen angibt. Diese Abstände müssen dann sowohl horizontal als auch vertikal definiert werden. Bei dem Applet ist dieser Standardwert verwendet worden. Ein Vorteil gegenüber dem GridLayout besteht darin, dass ein FlowLayout die gewünschte Größe eines Dialogelementes berücksichtigt, ein GridLayout passt die Größe der Dialogelemente der Größe des Fensters an. Falls man also die Größe des Fensters ändert, ändert sich auch die Größe Elementes. In [4] findet sich hierzu in dem Abschnitt 31.2 ein Beispiel. Die Abbildung 31.4 (siehe A.3) zeigt dort fünf Buttons angeordnet innerhalb eines 500*200 Pixel großen Fensters unter Verwendung eines FlowLayouts, Abbildung 31.5 (siehe A.4) zeigt sieben Buttons bei einem Fenster gleicher Größe unter Verwendung eines GridLayouts mit den Werten (4,2).

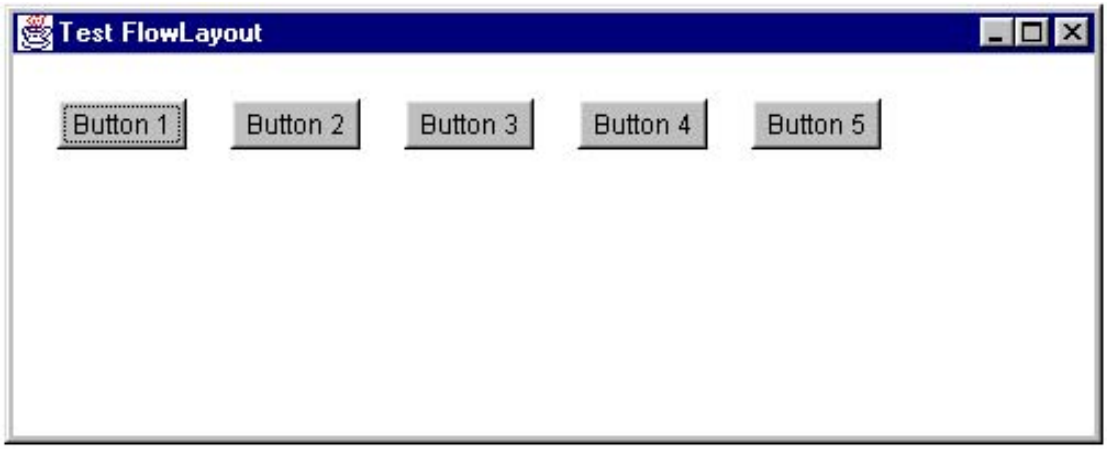

Abbildung A.3.: Buttondarstellung in einem FlowLayout entnommen aus [4],Abschnitt 31.2

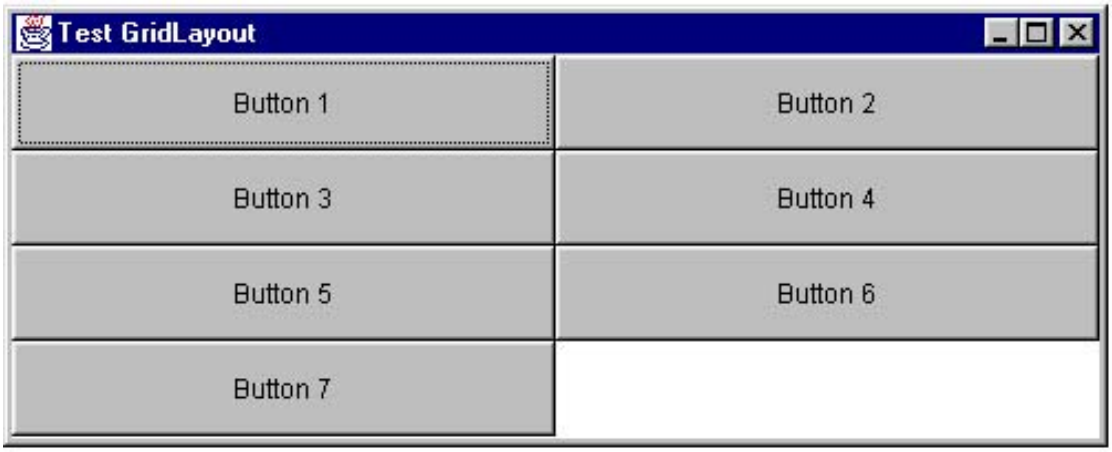

Abbildung A.4.: Buttondarstellung in einem GridLayout entnommen aus [4],Abschnitt 31.2

Das Panel verwendet also ein FlowLayout um einer solchen ungewollten Deformierung der Buttons entgegen zu wirken. Die Implementierung des dritten Panels sieht dementsprechend wie folgt aus:

Quellcodeausschnitt A.7 ()

```
p3.setLayout(new FlowLayout(FlowLayout.CENTER));
p3.add(leftButton);
p3.add(rightButton);
leftButton.addActionListener(this);
rightButton.addActionListener(this);
```

Führt man diesen Teil des Quellcodes aus erhält man folgenden Ausschnitt des Applets:

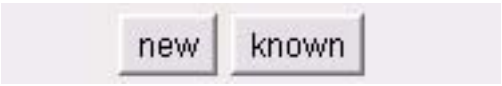

Abbildung A.5.: Abbildung des dritten Panels

Zusätzlich werden an dieser Stelle des Applets noch die beiden Buttons einem ActionListener hinzugefügt, welche im späteren Verlauf bei der Erfassung und Samm-

lung von Daten genauer behandelt werden.

A.4. Das main Panel

Nach der Implementierung der einzelnen Panels müssen diese noch zusammengesetzt werden um das gesamte Applet aufzubauen. Um dies zu ermöglichen wird ein viertes Panel initialisiert, welches die anderen drei Panels als Dialogelemente aufnimmt und entsprechend anordnet. Dadurch entsteht ein geschachteltes Layout. Da die Größe der einzelnen Panels dabei unterschiedlich sein soll und die Panels untereinander angeordnet werden sollen, kann man weder ein FlowLayout noch ein GridLayout verwenden um die einzelnen Panels zu positionieren. Bei diesem Panel wird deshalb ein GridBagLayout verwendet. Dieses Layout arbeitet ähnlich wie ein GridLayout, allerdings kann man bei einem GridBag die Position und die Ausmaße der einzelnen Dialogelemente selbst festlegen. Dementsprechend ist das Einfügen der Elemente aufwändiger, da man für jedes Element noch GridBagConstraints angeben muss, welche die individuellen Angaben für jedes einzelne Element anlegen. Bei dem Applet werden für jedes einzelne Element folgende Constraints angegeben:

gridx: Dieser Wert gibt die Anfangsspalte für das entsprechende Dialogelement an. An genau diesem Punkt ist die obere linke Ecke des Dialogelementes. Dabei hat die erste Spalte den Wert 0.

gridy: Dieser Wert gibt die Anfangszeile für das entsprechende Dialogelement an. Bei genau diesem Wert ist der obere Rand des Dialogelementes. Auch hier hat die erste Zeile den Wert 0. Fasst man beide Werte zusammen ergibt sich der Anfangspunkt (0,0) für das Gitternetz.

gridwith: Dieser Wert gibt die horizontale Ausdehnung des Dialogelementes an.

gridheight: Dieser Wert gibt die vertikale Ausdehnung des Dialogelementes an. Die jeweilige Ausdehnung bezieht sich auf die Spalten und Zeilen innerhalb des Gitters. Gibt man für die Werte eine entsprechende Zahl an, so wird das Element über die entsprechende Anzahl an Gitterfeldern ausgedehnt.

Die obigen Constraints werden innerhalb des Panels für jedes der einzelnen Panels angegeben. Innerhalb des Applets sieht die Implementierung des vierten Panels mit den entsprechenden Constraints wie folgt aus:

Quellcodeausschnitt A.8 ()

```
GridBagLayout main = new GridBagLayout();
GridBagConstraints gbc = new GridBagConstraints();
setLayout(main);
gbc.gridx = 0;
gbc.gridy = 0;
gbc.gridwidth = 1;
gbc.gridheight = 4;
main.setConstraints(p1, gbc);
add(p1);
gbc.gridx = 0;
gbc.gridy = 5;
gbc.gridwidth = 1;
gbc.gridheight = 1;
main.setConstraints(p2, gbc);
add(p2);
gbc.gridx = 0;
gbc.gridy = 6;
gbc.gridwidth = 1;
gbc.gridheight = 1;
main.setConstraints(p3, gbc);
add(p3);
```

Wie man aus dem Quelltext erkennen kann haben die entsprechenden Elemente der Panels die gleiche Breite, allerdings hat das erste Panel eine im Verhältnis größere vertikale Ausrichtung als die anderen beiden Elemente. Das Ergebnis dieses geschachtelten Layout - Managers ergibt die endgültige Benutzeroberfläche des Applets, welche wie folgt aussieht:

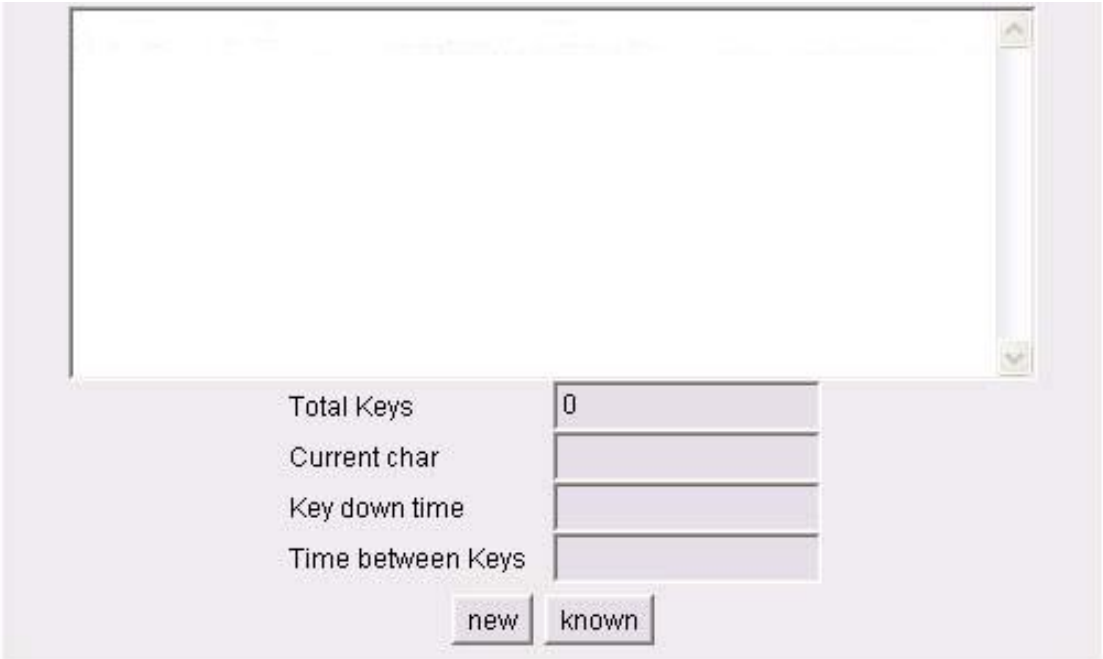

Abbildung A.6.: Grafische Oberfläche des vollständigen Applets

B. Quellcode

Dieser Abschnitt enthält den gesamten Quellcode des Java - Applets, allerdings ohne die einzelnen Komentare zu den jeweiligen Methoden mit einzubeziehen. Um die Navigation zu vereinfachen ist der Quellcode in einzelne Methodenabschnitte unterteilt.

B.1. imports

```
import java.awt.*; // importing Java Application window tool package
import java.applet.*; //importing Java applet package
import java.awt.event.*; // importing Java.awt.event - Managing Package
import java.io.*; // imoprting Java Input -Output Package
```

B.2. Die Klasse Keystroke

```
public class Keystroke
extends Applet
implements ActionListener, KeyListener
{
float[] extension = new float[9];
long [][] bksave = new long [3][2000];
long [][] kdsave = new long [3][2000];
int [][] keycodes = new int [3][2000];
Button leftButton = new Button("new");
Button rightButton = new Button("known");
```

```
public TextArea text = new TextArea("",10,60,1);
public TextField tk = new TextField("0");
public TextField cc = new TextField();
public TextField kdt = new TextField();
public TextField tbk = new TextField();
Panel p1 = new Panel();
Panel p2 = new Panel();
Panel p3 = new Panel();
int secdimensioncount = 0, firstdimensioncount = 0;
int differ = 45;
long timedown = 0, timerelease = 0;
```

B.3. die init - Methode

```
public void init()
{
}
```

B.4. Die Methode Keystroke

```
public Keystroke ()
{
Color c = new Color(0xFFE9E9E9);
tk.setEditable(false); // bars the User from Editing within
cc.setEditable(false); // the Textfields used in the
kdt.setEditable(false);// second Panel
tbk.setEditable(false);
p1.setLayout(new GridLayout(1,1)); // Setting the first Panel
p1.add(text); // Adding Textelement to Panel 1
text.append("If you need to create sample values, please push the");
text.append("Button 'new', otherwise chose the Button 'known'");
text.addKeyListener(this); // adding KeyListener
p2.setLayout(new GridLayout(4,2)); // Setting the second Panel
```

```
p2.add(new Label("Total Keys"));
p2.add(tk); // Adding Labels an corresponding TextFields
p2.add(new Label("Current char"));
p2.add(cc);
p2.add(new Label("Key down time"));
p2.add(kdt);
p2.add(new Label("Time between Keys"));
p2.add(tbk);
p3.setLayout(new FlowLayout(FlowLayout.CENTER)); // Setting the third Panel
p3.add(leftButton); // Adding Buttons
p3.add(rightButton);
leftButton.addActionListener(this); // Adding ActionListener
rightButton.addActionListener(this);
GridBagLayout main = new GridBagLayout(); // instancing main Panel
GridBagConstraints gbc = new GridBagConstraints();
setLayout(main);// Setting mains Panel and adding the three Panels
gbc.gridx = 0; // with individual Settings to the main Panel
gbc.gridy = 0;
gbc.gridwidth = 1;
gbc.gridheight = 4;
main.setConstraints(p1, gbc);
add(p1);
gbc.gridx = 0;
gbc.gridy = 5;
gbc.gridwidth = 1;
gbc.gridheight = 1;
main.setConstraints(p2, gbc);
add(p2);
gbc.gridx = 0;
gbc.gridy = 6;
gbc.gridwidth = 1;
gbc.gridheight = 1;
main.setConstraints(p3, gbc);
add(p3);
setBackground(c); // Setting Backgrond of main Panel
}
```

B.5. Die Methode actionPerformed

```
public void actionPerformed(ActionEvent e)
{
        String Input = e.getActionCommand();
if (Input.equals("Clear")) // resetting all values
{
for(int x = 0; x <=secdimensioncount; x++)
{ // deleting all values captured till clear was pushed
keycodes[firstdimensioncount][x] = 0;
bksave[firstdimensioncount][x] = 0;
kdsave[firstdimensioncount][x] = 0;
}
text.setText(""); // resetting TextFields
tk.setText("0");
cc.setText("");
kdt.setText("");
tbk.setText("");
secdimensioncount = 0; // resetting Counter
}
else if(Input.equals("Step2")) // enter second Input Step
{
firstdimensioncount ++; // increment first dimension for the Arrays
text.setText(""); // resetting TeftFields
tk.setText("0");
cc.setText("");
kdt.setText("");
tbk.setText("");
secdimensioncount = 0; // resetting counter
rightButton.setLabel("1.result"); // Set right Button to 1.result
}
else if(Input.equals("1.result"))// entering first evaluation
{
leftButton.setLabel("--");// freezes Left Button
text.setEditable(false); // Disable Editing for TextArea
text.setText("Starting Computation\n");
average(); // Execute method average()
```

```
evaluations(); // Execute method evaluations()
Datasave(); // Save captured values
tk.setText("0"); // reset Textfields
cc.setText("");
kdt.setText("");
tbk.setText("");
firstdimensioncount ++; // increment First Arraydimension
text.append("Computation completed!\n");
text.append("If you want to proceed now push 'Step3', ");
text.append("otherwise push 'Close'!");
leftButton.setLabel("Close"); // Sets choice to go on directly
rightButton.setLabel("Step3");// or go on later
}
else if(Input.equals("Step3"))// enter the third Input Step
{
text.setText(""); // reset TextArea
leftButton.setLabel("Clear"); // Set left Button to Clear
text.setEditable(true); // allow user to Edit Text
secdimensioncount = 0; // reset second Arraydimension
rightButton.setLabel("Done"); // set right Button to Done
}
else if(Input.equals("Done")) // enter final comparison
{
leftButton.setLabel("--"); // freeze the left Button
text.setEditable(false); // Disable Editing for TextArea
text.setText("Starting Comparison\n\n\n");
compare(); // Executing method compare
text.append("FINISHED");
leftButton.setLabel("Close"); // set left Button to Close
}
else if(Input.equals("Close")) // shows Final message to close Applet
{
text.setText("Thank you for your Participation!!\n");
text.append("Please close browser!");
destroy();

}
else if (Input.equals("--")) // Freezes a Button
```

```
{
}
else if (Input.equals("new"))// starts Applet at the beginning in order
{ // create new sample values
leftButton.setLabel("Clear"); // set left Button to Clear
rightButton.setLabel("Step2");// set right Button to Step2
text.setText(""); // reset Textarea
}
else if (Input.equals("known")) // starts Applet at the third Input Step
{ // in order to compare values
Dataread(); // reading Data
leftButton.setLabel("--"); // freezing left Button
rightButton.setLabel("Step3"); // set right Button to Step3
text.setText("Values read: \n");
for(int x = 0; x< extension.length; x++) // print out values read
{
text.append ( (x+1)+" value read = "+extension[x]+" !\n");
}
text.append("Push Step3 to proceed!\n");
}
}
```

B.6. Die Methode keyPressed

```
public void keyPressed(KeyEvent e)
{
int key=0;
String s, ss;
long Betweenkey;
key = (e.getKeyCode()); // capturing keycode
keycodes[firstdimensioncount][secdimensioncount] = key; // saving
s =(new Character(e.getKeyChar())).toString(); // converting code to String
cc.setText(s);
timedown = System.currentTimeMillis(); // Capturing timestamp
Betweenkey = timedown - timerelease; // evluating tbk
```

```
ss = (new Long(Betweenkey)).toString(); // converting long to String
tbk.setText(ss);
bksave[firstdimensioncount][secdimensioncount] = Betweenkey; // saving
}
```

B.7. Die Methode keyReleased

```
public void keyReleased(KeyEvent e)
{
long Keydown;
String s,ss;
timerelease = System.currentTimeMillis();//capturing timestamp
Keydown = timerelease - timedown; //evaluating kd
s = (new Long(Keydown)).toString(); // converting long to String
kdt.setText(s);
kdsave[firstdimensioncount][secdimensioncount] = Keydown; // saving
secdimensioncount++; // increment second Dimension
ss = (new Integer(secdimensioncount)).toString(); // convert int to String
tk.setText(ss);
}
```

B.8. Die Methode keyTyped

```
public void keyTyped(KeyEvent e)
{
}
```

B.9. Die Methode average

```
public void average()
{
```

```
float [] result = new float[2];
// start computing average bksave
for (int x=1; x<=2; x++) // initialize outer loop
{
long mention = 0; // local variable for adding values
int count = 0; // local count variable
text.append("Computing average Time between keys for the "+x+". Step!\n ");
text.append("Captured Values:\n");
for(int y=1 ; y<bksave[x-1].length; y++) // init inner loop
{
if (bksave[x-1][y] != 0) //check if value can be mentioned
{
tbk.setText(""+bksave[x-1][y]+"\n");
mention += bksave[x-1][y]; // adding captured values
count++; //increment count
}
}
text.append("Added times for this Step "+mention +"!\n");
text.append(count+" time(s) counted\n");
text.append("Average Time for this Step: "+ mention/count+"\n");
result[x-1] = mention/count; // saving interim value
}
extension[0] = (result[0]+result[1])/2; // evaluating and saving final value
text.append("The final average Time between the Keys is "+extension[0]+"\n");
// start copmuting average kdsave
for (int x=1; x<=2; x++) // init outer loop
{
long mention = 0; //local Value for adding times
int count = 0; // local count Value
text.append("Computing average Key down time for the "+x+". Step!\n");
text.append("Captured Values:\n");
for(int y=1 ; y<kdsave[x-1].length; y++) // inti inner loop
{
if (kdsave[x-1][y] != 0) // check if value can be mentioned
{
kdt.setText(""+kdsave[x-1][y]+"\n");
mention += kdsave[x-1][y]; // Adding captured values
count++; //increment count
```

```
}
}
text.append("Added times for this Step "+mention +"\n");
text.append(count+" time(s) counted\n");
text.append("Average time for this Step: "+ mention/count+"\n");
result[x-1] = mention/count; //saving interim value
}
extension[1] = (result[0]+result[1])/2; // evaluating and saving final value
text.append("The final average for the Key down time is ");
text.append(extension[1]+"\n");
//start computing average keycodes
for (int x=1; x<=2; x++) //init outer loop
{
int count = 0; // local count variable
text.append("Counting chars for the "+x+". Step\n");
for(int y=0; y<keycodes[x-1].length; y++)
{
if (keycodes[x-1][y]!=0) // check if value can be mentioned
{
tk.setText(""+keycodes[x-1][y]+"\n");
count++; // increment count
}
}
text.append(count+" chars typed in the "+x+". Step!\n");
result[x-1]= count; //saving interim value
}
extension[2]=(result[0]+result[1])/2; // saving and evaluating final value
text.append("Average use of chars in both steps is "+extension[2]+"\n");
}
```

B.10. Die Methode evaluations

```
public void evaluations()
{
float [] result = new float[6];
```

```
// start evaluating the use of Backspaces within an input step
text.append("Evaluating use of Backspaces\n");
for(int x=1;x<=2;x++) // init outer loop
{
int count=0; // Local count variable
int count2 = 0; // local count variable
text.append("Counting use of Backspace in the "+x+". Step:\n");
if (keycodes[x-1][(keycodes[x-1].length-1)]==8) // check backspace located at endpos
{
text.append("Backspace found!\n");
count ++;
}
for(int y=1; y<(keycodes[x-1].length-1); y++) // init inner loop
{ // chek if two following backspaces can be detected
if (keycodes[x-1][y]==8 && keycodes[x-1][y+1]==8)
{
text.append("two Backspaces found at the positons");
text.append((y-1)+" and "+y+" !\n");
count2 ++; // increment count2
y++;
} // else check if one backslash can be detected
else if (keycodes[x-1][y]==8)
{
text.append("Backspace found at Position "+y+"!\n");
count++; // increment count
}
}
text.append(count+" Backspaces found in the "+x+". Step!\n");
result[x-1] = count; // save interim value for one Backspace
text.append(count2+" two Backspaces found in the "+x+". Step!\n");
result[x+1] = count2; // save interim value for 2 Backspaces
} // evaluate and save final values for the counted Backspaces
extension[3]=(result[0]+result[1])/2;
text.append("Found Backspaces in Average: "+extension[3]+"!\n");
extension[4]=(result[2]+result[3])/2;
text.append("Found two Backspaces in Average "+extension[4]+"!\n");
// start working on the Array kdsave
text.append("Calculating Differences in the time sets.\n");
```

```
for(int x=1;x<=2;x++) // init outer loop
{
int count = 0; // local count variable
text.append("Evaluating Differences for the key down time in the ");
text.append(x+". Step.\n");
for(int y=1; y<kdsave[x-1].length; y++) // init inner loop
{
if (kdsave[x-1][y]!=0) // check if Vale can be mentioned
{  // check if Value differs mor than differ ms from average
if((kdsave[x-1][y]< (extension[1]-differ))||
       (kdsave[x-1][y]> (extension[1]+differ)))
{
count++; // increment count
text.append("Difference found at Position "+y+"!\n");
}
}
}
result[x-1] = count; // saving interim value
text.append("The Result for the "+x+". Step is "+count+".\n");
} // evaluating and saving final value
extension[5] = (result[0] + result[1])/2;
text.append("Average Result for both steps: " +extension[5]+"\n");
// start working on the Array bksave
for(int x=1;x<=2;x++) // init outer loop
{
int count = 0; // local count vraibles
int count2 = 0;
int count3 = 0;
text.append("Evaluating Differences for the time between keys for ");
text.append("the "+x+". Step.\n");
for(int y=1; y<bksave[x-1].length; y++) // init inner loop
{  // chek if current value differs more thann 200ms from average
if(bksave[x-1][y]>(extension[0]+200))
{
count++; // increment count
text.append("Difference greater than 200 units found at ");
text.append("position "+y+"!\n");
} // else check if value differs more than 100ms from average
```

```
else if(bksave[x-1][y]>(extension[0]+100))
{
count2++; // increment count2
text.append("Difference greater than 100 units found at ");
text.append("position "+y+"!\n");
} // else check if value differs more than 50ms from average
else if(bksave[x-1][y]>(extension[0]+50))
{
count3++; // increment count3
text.append("Difference greater than 50 units found at ");
text.append("position "+y+"\n");
} // saving interim values
result[x-1] = count;
result[x+1] = count2;
result[x+3] = count3;
}
text.append(result[x-1]+" differences greater than 200, \n");
text.append(result[x+1]+" differences greater than 100 and \n");
text.append(result[x+3]+" differences greater than 50 evaluated in");
text.append(" the "+x+". Step.\n");
} // evaluating and saving final values
extension[6] =(result[0] + result[1]) /2;
extension[7] =(result[2] + result[3]) /2;
extension[8] =(result[4] + result[5]) /2;
text.append(extension[6]+" average differences greater than 200, \n");
text.append(extension[7]+" average differences greater than 100 and\n");
text.append(extension[8]+" average differences greater than 50 evaluated.\n");
}
```

B.11. Die Methode compare

```
public void compare ()
{
float match = 0;
long result[] = new long [2];
```

```
long mention = 0;
int count = 0, count2 = 0, count3 = 0;
text.append("Calculating time between Keys for the 3. Step.\n");
text.append("Captured values: \n");
// start evaluating and comparing average bksave
for (int x = 1; x<bksave[firstdimensioncount].length;x++) // init loop
{ // check if value can be mentioned
if (bksave[firstdimensioncount][x]!= 0)
{
text.append(bksave[firstdimensioncount][x]+"\n");
mention += bksave[firstdimensioncount][x]; // Adding values
count++; // increment count
}
} // evaluating and saving result
result[0] = mention/count;
text.append("Added times for this Step: "+mention+"!\n");
text.append(count+" time(s) counted!");
text.append("Average Time between Keys in the 3. Step: ");
text.append(result[0]+".\n");
// comparing sample and estimated value and printing out difference
if (result[0]>extension[0])
{
text.append("Difference between sample time and evaluated time");
text.append(" is:"+(result[0]-extension[0])+"!\n" );
match += (extension[0]/result[0]);
}
else
{
text.append("Difference between sample time and evaluated time");
text.append(" is:"+(extension[0]-result[0])+"!\n" );
match += (result[0]/extension[0]);
}
count = 0; // resetting used variables
mention = 0;
// start computing average kdsave
text.append("Calculating Key down time for the 3. Step.\n");
text.append("Captured Values: \n");
for (int x = 1; x<kdsave[firstdimensioncount].length;x++)//init loop
```

```
{ // check if current value can be mentioned
if (kdsave[firstdimensioncount][x]!= 0)
{
text.append(kdsave[firstdimensioncount][x]+"\n");
mention += kdsave[firstdimensioncount][x];// adding values
count++; //increment count
}
} // evaluate and save interim value
result[1] = mention/count;
text.append("Added times for this Step: "+mention+"!\n");
text.append(count+" time(s) counted!");
text.append("Average Key down Time in the 3. Step: ");
text.append(result[1]+".\n");
// comparing sample and estimated value and printing out difference
if (result[1]>extension[1])
{
text.append("Difference between sample time and evaluated time");
text.append(" is:"+(result[1]-extension[1])+"!\n" );
match += (extension[1]/result[1]);
}
else
{
text.append("Difference between sample time and evaluated time");
text.append(" is:"+(extension[1]-result[1])+"!\n" );
match += (result[1]/extension[1]);
}
mention = 0; // resetting values
count = 0;
// start computing average keycodes
text.append("Counting chars for the 3. Step");
for (int x = 0; x<keycodes[firstdimensioncount].length; x++)// init loop
{ //check if value should be mentioned
if (keycodes[firstdimensioncount][x]!= 0 )
{
text.append(keycodes[firstdimensioncount][x]+"\n");
count++; // increment count
}
} //comparing sample and estimated value and printing out difference
```

```
text.append(count+" chars counted in the 3. Step!\n");
if (count>extension[2])
{
text.append("Difference between sample value and evaluated value");
text.append(" is:"+(count-extension[2])+"!\n");
match += (extension[2]/count);
}
else
{
text.append("Difference between sample value and evaluated value");
text.append(" is:"+(extension[2]-count)+"!\n");
match += (count/extension[2]);
}
count = 0; // reset values
// start computing use of Backspaces
text.append("Evaluating use of Backspaces\n");
if (keycodes[firstdimensioncount][keycodes[firstdimensioncount].length-1] == 8) // c
{ // at endposition
text.append("Backspace found at position 0\n");
count++; // increment count
}
for (int x = 1; x<(keycodes[firstdimensioncount].length-1); x++)//init loop
{ // check if two backspaces are detected
if ((keycodes[firstdimensioncount][x] == 8) &&
(keycodes[firstdimensioncount][x+1] == 8))
{
text.append("2 Backspaces found");
count2++; // increment coun2
x++;
} // else check if one Backspace is detected
else if (keycodes[firstdimensioncount][x] == 8)
{
text.append("Backspace found at position"+x+"\n");
count++; // increment count
}

}
text.append("evaluated Backspaces in the 3. Step:"+count+"\n");
```

```
text.append("evaluated 2 Backspaces in the 3. Step:"+count2+"\n");
//comparing sample and estimated value and printing out difference
if (count>extension[3])
{
text.append("Difference between sample Backspaces and evaluated ");
text.append("Backspaces is:"+(count - extension[3])+"!\n");
match += (extension[3]/count);
}
else
{
text.append("Difference between sample Backspaces and evaluated ");
text.append("Backspaces is:"+(extension[3] - count)+"!\n");
match += (count/extension[3]);
}
if (count2>extension[4])
{
text.append("Difference between sample of 2 Backspaces and ");
text.append("evaluated Backspaces is:"+(count2 - extension[4])+"!\n");
match += (extension[4]/count);
}
else
{
text.append("Difference between sample of 2 Backspaces and  ");
text.append("evaluated Backspaces is:"+(extension[4] - count2)+"!\n");
match += (count/extension[4]);
}
count = 0; // resetting values
count2 = 0;
// start working on the Array kdsave
text.append("Evaluating Differences in the Key down time in the 3. Step\n");
for (int x = 1; x<kdsave[firstdimensioncount].length; x++) // init loop
{ // Check if Value can be mentioned
if (kdsave[firstdimensioncount][x]!=0)
{ //check if current value differs from average more than differ
if((kdsave[firstdimensioncount][x]< (result[1]-differ))||
       (kdsave[firstdimensioncount][x]> (result[1]+differ)))
{
count++; // increment count
```

```
text.append("Difference found at Position "+x+"!\n");
}
}
}
text.append(count+" Differences found in the 3. Step!\n");
//comparing sample and estimated value and printing out difference
if (count>extension[5])
{
text.append("Difference between sample value and evaluated value ");
text.append("is "+(count-extension[5])+"!\n");
match += (extension[5]/count);
}
else
{
text.append("Difference between sample value and evaluated value ");
text.append("is "+(extension[5]-count)+"!\n");
match += (count/extension[5]);
}
count = 0 ; // reset variable
// start working on the Array kdsave
text.append("Evaluating differences for the time between keys in the 3. Step\n");
for(int x = 1; x<bksave[firstdimensioncount][x]; x++) // init loop
{ // check if current value differs more than 200ms from average
if(bksave[firstdimensioncount][x]>(result[0]+200))
{
count++; // increment count
text.append("Difference greater than 200 units found at ");
text.append("position "+x+"!\n");
} //else chek if current value differs more than 100ms from average
else if(bksave[firstdimensioncount][x]>(result[0]+100))
{
count2++; //increment count2
text.append("Difference greater than 100 units found at ");
text.append("position "+x+"!\n");
} //else check if current value differs more than 50ms from average
else if(bksave[firstdimensioncount][x]>(result[0]+50))
{
count3++; //increment count3
```

```
text.append("Difference greater than 50 units found at ");
text.append("position "+x+"\n");
}

}
text.append(count+" differences greater than 200, \n");
text.append(count2+" differences greater than 100 and \n");
text.append(count3+" differences greater than 50 evaluated in");
text.append(" the 3. Step.\n");
//comparing sample and estimated value and printing out difference
if(count>extension[6])
{
text.append("Difference between sample value and evaluated value");
text.append(" for 200 units is:"+(count-extension[6])+"!\n");
match += (extension[6]/count);
}
else
{
text.append("Difference between sample value and evaluated value");
text.append(" for 200 is:"+(extension[6]-count)+"!\n");
match += (count/extension[6]);
}
if(count2>extension[7])
{
text.append("Difference between sample value and evaluated value");
text.append(" for 100 units is:"+(count2-extension[7])+"!\n");
match += (extension[7]/count);
}
else
{
text.append("Difference between sample value and evaluated value");
text.append(" for 100 units is:"+(extension[8]-count2)+"!\n");
match += (count/extension[7]);
}
if(count3>extension[8])
{
text.append("Difference between sample value and evaluated value");
text.append(" for 50 units is:"+(count3-extension[8])+"!\n");
```

```
match += (extension[8]/count);
}
else
{
text.append("Difference between sample value and evaluated value");
text.append(" for 50 units is:"+(extension[8]-count3)+"!\n");
match += (count/extension[8]);
}
match = match /9;
if(match>=0.95)
{
text.append("\n\n\n Matching Score is "+match+"! ");
text.append("Within a range of difference of 5% the User ist accepted!");
}
else
{
text.append("\n\n\n Matching Score is "+match+"! ");
text.append("Within a range of difference of 5% the User ist not accepted!");
}
leftButton.setLabel("Close");
}
```

B.12. Die Methode Datasave

```
public void Datasave()
{
try // init try block in order to catch exception that may be thrown
{ // needs to be implemented when working with an Output sequence
DataOutputStream out = new DataOutputStream // init DataOutputStream
    ( new BufferedOutputStream
      ( new FileOutputStream ("biodata.txt") ) );
     for(int x = 0; x<extension.length; x++) // init loop
     {  // Writing Data in Buffer
     out.writeFloat(extension[x]);
     text.append("Data exporting\n");
```

```
     }
     out.flush(); // empty Buffer --> Writing Data in File
     out.close(); // closing DataOuputStream
     }  // catching exceptions tha may be caused by the Output Sequence
     catch(FileNotFoundException e)
     {
     text.append("-- File biodata.txt not found --");
     }
     catch(IOException e)
     {
     text.append("-- Unable to write Data! --");
     }
}
```

B.13. Die Methode Dataread

```
public void Dataread()
{
try // init try block in order to catch exception that may be thrown
{ // needs to be implemented when working with an Input sequence
DataInputStream in = new DataInputStream //init DataInputStream
(new FileInputStream("biodata.txt"));
for(int x = 0; x<extension.length; x++) // init loop
{ //reading Data
extension[x] = in.readFloat();
}
in.close(); // closing DataInputStream
} // catching exceptions tha may be caused by the Input Sequence
catch(FileNotFoundException e)
{
text.append("---File biodata.txt not found ---\n");
text.append("New sample Values need to be created!\n");
text.append("Please restart Applet!");
}
catch(IOException e)
```

```
{
text.append("---Unable to read Data ---\n");
text.append("New sample Values need to be created!\n");
text.append("Please restart Applet!");
}

}
}
```

C. Detaillierte Zertifikatserstellung

Die Zertifizierung

Führt man eine Speicherung aus, ohne vorher das Applet zertifiziert zu haben, so wird während der Laufzeit ein Fehler durch den Security Manager ausgelöst. Es muss ein entsprechendes Zertifikat an das Applet gebunden werden, so dass ein korrektes Ausführen möglich ist. Java bietet dafür eine Möglichkeit, allerdings muss man zuerst das Applet in ein jar - Archiv packen, da nur diese mit Hilfe der von Java bereitgestellten Werkzeuge zertifiziert werden können. Um ein jar - Archiv zu erstellen muss man zuerst eine Manifest - Datei erstellen. In dieser Datei stehen Informationen über die in dem Archiv enthaltenen Klassen. Vor allem wird angegeben, wo sich die Hauptklasse des gesamten Programms befindet. Wird eine solche Datei nicht selbst verfasst, erstellt das Programm *jar* automatisch eine Manifest - Datei, über die man aber dann dementsprechend keinen Einfluss hat. Die Manifest - Datei für das Applet hat folgendes Aussehen.

Quellcodeausschnitt C.1 ()

```
Manifest-Version: 1.0
MAIN-CLASS: Keystroke
```

Die erste Zeile gibt die verwendete Version des Manifests an, die zweite Zeile verweist auf die Klasse, welche die main - Methode, beziehungsweise in diesem Fall die init - Methode, enthält. Um sicherzustellen, dass bei der Verarbeitung auch alle angegebenen Werte berücksichtigt werden, muss nach dem letzten angegebenen Parameter in der Datei noch eine freie Zeile stehen, da ansonsten das Programm die letzte Zeile nicht einliest. Nun kann man mit Hilfe der class - Datei, welche bei dem Kompilieren des Quellcodes gebildet wird und der Manifest - Datei ein jar - Archiv generieren. Mit dem Aufruf des Programms *jar* erhält man eine Auflistung

der möglichen Optionen, die bei dem Erstellen eines jar - Archivs zur Verfügung gestellt werden. Im Folgenden werden die Optionen kurz erklärt, die von mir verwendet worden sind:

c: Diese Option gibt an, dass ein neues Archiv erstellt werden soll.

v: Durch diese Option wird eine genauere Ausgabe bei der Generierung des Archivs angefordert. Diese Option ist nicht zwingend notwendig, wurde aber angewandt um genau sehen zu können, dass die richtige Datei eingebunden worden ist und um Informationen über den Grad der Komprimierung zu erhalten.

f: Mit Hilfe dieser Option kann man genau den Namen des entstehenden Archivs spezifizieren. Dabei ist darauf zu achten, dass der Name auch die Endung *.jar* trägt.

m: Durch diese Option werden bei dem Generieren die Daten, welche innerhalb der Manifest - Datei abgelegt sind verwendet anstatt eine default - Datei zu verwenden.

Bei dem eigentlichen Befehl zur Erstellung ist dringend darauf zu achten die Reihenfolge der einzelnen Eingabedateien entsprechend der Reihenfolge der Optionen anzugleichen, da ansonsten Fehler auftreten. Für das Applet sieht der vollständige Befehl zur Erstellung wie folgt aus:

Quellcodeausschnitt C.2 ()
```
jar cvfm Keystroke.jar Manifest.txt Keystroke.class
```

Um das Speichern zu ermöglichen muss nun ein Zertifikat erstellt werden. Dazu muss zuerst ein public - private - Key Paar erstellt werden, welches für die Erstellung eines Zertifikates benötigt wird. Dieses Schlüsselpaar kann man über den Befehl *genkey* innerhalb des keytool - Werkzeugs erstellen. Optional kann man noch einen entsprechenden Namen angeben, unter dem das Schlüsselpaar geführt wird. Bei dem Aufruf des Befehls verlangt das Programm einige Angaben, bevor das entsprechende Schlüsselpaar erstellt wird. Für das Applet wurden dabei folgende Angaben gemacht:

Quellcodeausschnitt C.3 ()

```
C:\\Programme\\Java\\jdk1.5.0\_09\\bin>keytool -genkey -alias Torsten
Geben Sie das Keystore-Passwort ein:  stroke
Wie lautet Ihr Vor- und Nachname?
 [Unknown]:  Torsten Hermes
Wie lautet der Name Ihrer organisatorischen Einheit?
 [Unknown]:  fb 4
Wie lautet der Name Ihrer Organisation?
 [Unknown]:  Universität Koblenz
Wie lautet der Name Ihrer Stadt oder Gemeinde?
 [Unknown]:  Koblenz
Wie lautet der Name Ihres Bundeslandes oder Ihrer Provinz?
 [Unknown]:  Rheinland Pfalz
Wie lautet der Landescode (zwei Buchstaben) f^3r diese Einheit?
 [Unknown]:  RP
Ist CN=Torsten Hermes, OU=fb 4, O=Universität Koblenz, L=Koblenz, ST=Rheinland P
falz, C=RP richtig?
 [Nein]:  Ja
Geben Sie das Passwort f^3r < Torsten > ein.
        (EINGABETASTE, wenn Passwort dasselbe wie f^3r Keystore):
```

Nachdem die Angaben gemacht worden sind verlangt das Programm noch nach einem Passwort, welches eingegeben werden muss um das entsprechende Schlüsselpaar anwenden zu können. Will man kein zweites Passwort verwenden, so wird das Keystore - Passwort verwendet. Nachdem dieses Passwort eingegeben worden ist wird das Schlüsselpaar erstellt und in einem Keystore - Ordner abgelegt. Aufgerufen wird das Schlüsselpaar unter dem entsprechenden Alias „Torsten“, so wie er bei dem Aufruf des Befehls angegeben ist. Mit Hilfe des Schlüsselpaares kann man nun unter Verwendung des gleichen Werkzeugs ein Zertifikat für das Schlüsselpaar erstellen. Dies geschieht mit dem Befehl *selfcert*. Da bei der Erstellung des Schlüsselpaares kein default Wert als Alias verwendet worden ist, muss dieser bei der Zertifikatserstellung ebenfalls angegeben werden. Daher ergibt sich folgender Befehl:

Quellcodeausschnitt C.4 ()

```
C:\\Programme\\Java\\jdk1.5.0\_09\\bin> keytool -selfcert -alias Torsten
Geben Sie das Keystore-Passwort ein:  stroke
```

Dieses Zertifikat muss nun noch in das Archiv eingebunden werden. Dies geschieht mit dem Programm *jarsigner*, welches ein bestehendes Zertifikat mit einem Archiv koppeln kann. Um dies zu ermöglichen wird in dem Programm *jarsigner* der Befehl *-signedjar* aufgerufen. Da das zertifizierte Archiv sich von dem normalen Archiv unterscheiden soll, wird zusätzlich vor dem zu zertifizierenden Archiv noch ein neues Archiv angegeben, welches nach der Zertifizierung das Zertifikat und das Applet enthalten soll. Des Weiteren enthält der Befehlsaufruf noch die Alias des verwendeten Schlüsselpaares, mit dessen Hilfe das Zertifikat erstellt worden ist. Daher sieht der gesamte Befehl wie folgt aus:

Quellcodeausschnitt C.5 ()

```
C:\\Programme\\Java\\jdk1.5.0\_09\\bin> jarsigner -signedjar
skeystroke.jar keystroke.jar Torsten
Enter Passphrase for keystore: stroke
```

Nachdem erneut das Passwort für das Schlüsselpaar eingegeben worden ist wird das neue Archiv *skeystroke.jar* erstellt, welches nun das zertifizierte Applet beinhaltet. Durch die Verwendung dieses Archivs anstelle der class - Datei ist nun das Speichern der Daten möglich. Will man nun noch etwas an dem Applet ändern, so muss man jedes Mal erneut das Applet in ein Archiv packen und dieses zertifizieren, da das Zertifikat immer nur auf ein bestimmtes Archiv angewendet werden kann. Ändert man etwas, so verliert das Zertifikat seine Gültigkeit. Ein weiterer Nachteil ergibt sich durch die temporäre Dauer des Zertifikates. Der Vorgang der Zertifizierung muss alle sechs Monate wiederholt werden, da das Zertifikat dann seine Gültigkeit verliert [6, 7].

Nun ist das Applet in der Lage Daten auf einer Festplatte abzuspeichern, da der Benutzer bei dem Aufruf des Applets zuerst das selbst erstellte Zertifikat (siehe Abb. D.2) bestätigen muss bevor das eigentliche Applet gestartet wird.

D. Grafiken

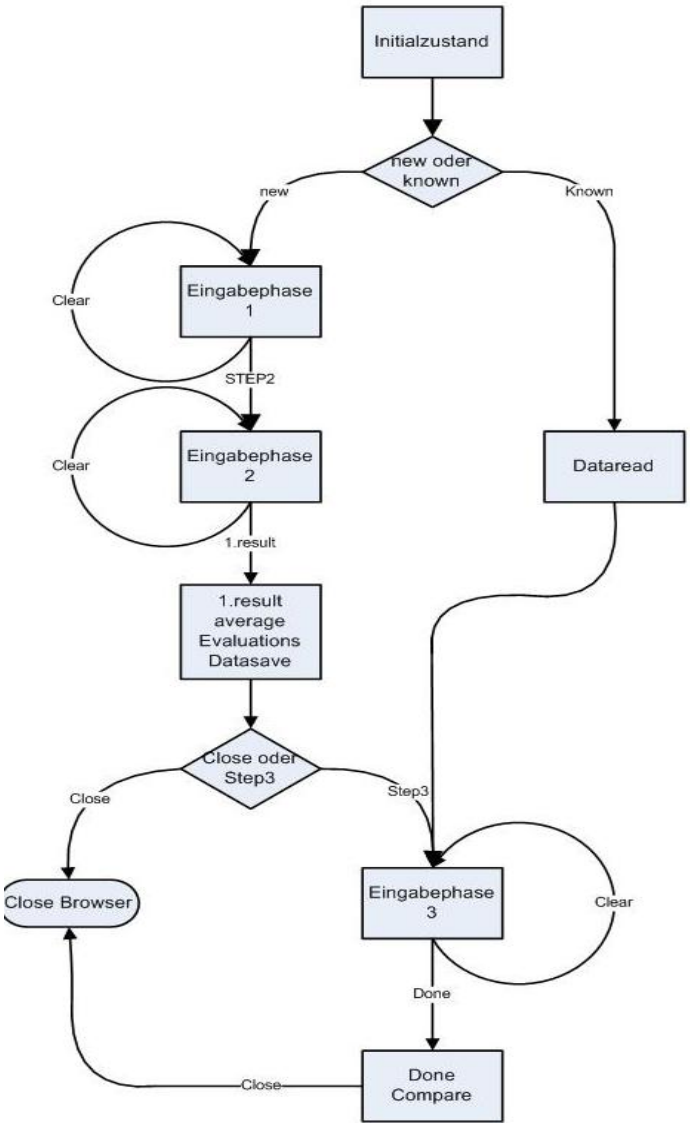

Abbildung D.1.: Ablaufdiagramm des Applets

Abbildung D.2.: Bild des Zertifikates, welches bestätigt werden muss um mit dem Applet arbeiten zu können