INSTITUT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Implementierung eines Werkzeuges zur graphischen Darstellung von WWW-Link-Strukturen

Diana Stricker

Albert Euba

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Alexander Keller
Abgabedatum:	26. Juli 1996

Inhaltsverzeichnis

Kapitel 1

Einführung

War noch vor einigen Jahren das Internet ein Tummelplatz für einige wenige, meist Hochschulangehörige, so ist es gegenwärtig im Begriff, zu einem Medium für jedermann zu werden. Den größten Anteil an dieser Entwicklung hat mit Sicherheit das World Wide Web (WWW) beigetragen, das mit seinen Möglichkeiten, Daten aller Art einfach und ansprechend zu präsentieren und miteinander zu verknüpfen, allen bisherigen Diensten im Internet in punkto Bedienungsfreundlichkeit weit überlegen ist und damit auf eine wesentlich höhere Akzeptanz in allen Anwenderkreisen stößt. Eine stark ansteigende Zahl von Benutzern, ist aber immer verbunden mit einer deutlichen Steigerung des Verwaltungsaufwandes. Deshalb werden neue Hilfsmittel benötigt, um diesen Verwaltungsaufwand zeitlich zu verringern. Dieses Fortgeschrittenenpraktikum soll als ein solches Hilfsmittel ein Werkzeug implementieren, das Link-Strukturen in geeigneter Weise graphisch darstellt.

1.1 Motivation

Durch den steigenden Verbreitungsgrad und die Bedienungsfreundlichkeit wird das WWW zunehmend auch für Firmen interessant. Die genannten Verknüpfungsmöglichkeiten bergen aber gewisse Hindernisse: Entfernt jemand eine von ihm angebotene WWW - Seite, so weiß er nicht, wer alles auf diese Seite verwiesen hat, wen er davon benachrichtigen müßte, daß sein Verweis (Link) ins Leere zeigt. Für Firmen, die sich im World Wide Web präsentieren, ist ein solcher fehlerhafter Verweis zwangsweise auch mit einem gewissen Imageverlust – die Firma xy kann ja nicht einmal ihre WWW-Seiten fehlerfrei anbieten; wie wird das erst mit dem Produkt sein? – verbunden und damit höchst unerwünscht. Die manuelle Kontrolle von Inkonsistenzen bei Verweisen ist jedoch mit erheblichem Zeitaufwand verbunden, vor allem, wenn die Anzahl der Dokumente sehr groß wird. Das kann aber auch automatisiert werden, was mit einem anderen Fortgeschrittenenpraktikum bereits geschehen ist. Dabei werden die Ergebnisse in einer Datei abgespeichert, aus der die gewünschten Daten ermittelt werden können.

Dieses Fortgeschrittenenpraktikum soll die Ergebnisse einer automatischen Konsistenzprüfung in geeigneter Weise graphisch veranschaulichen und damit leichter kontrollierbar machen. Solange es sich nur um relativ wenige Dokumente handelt, kann auch eine textorientierte Präsentation die Aufgabe bestens erfüllen. Mit zunehmender Größe der Strukturen hat eine Visualisierung aber mehr Vorteile. Wenn die Gesamtstruktur dargestellt wird und dabei fehlerhafte Verweise farblich abgehoben werden, sind sie bereits auf der ersten Blick zu erkennen. Genauere Informationen aus der Flut von Daten will man in den meisten Fällen nur über einzelne Dokumente erfahren. Diese aus einer graphischen Darstellung per Mausklick auszuwählen, ist deutlich unkomplizierter, als endlose Listen zu durchstöbern. Falls schließlich die gewählte HTML - Struktur zu groß ist, um auf einen Blick überschaubar zu sein, sollte auch eine Zoom–Funktion berücksichtigt werden.

1.2 Die Basis des Programms

Wie vorher schon erwähnt, wurde zur Konsistenzprüfung von HTML-Links bereits ein Programm – im Fortgeschrittenenpraktikum von Frank Schütz [Sch96] – implementiert. Dieses wird über eine `.ini`-Datei kontrolliert, in der die Start-URL sowie die zu durchsuchenden Server und einige andere Informationen angegeben werden müssen. Es prüft die so konfigurierte Struktur und erzeugt dabei drei Log-Dateien (siehe Abb. 1.1):

- *.ERL.LOG mit allen korrekt bearbeiteten Dokumenten
- *.ERR.LOG mit allen fehlerhaften (z.B. nicht erreichbaren) Dokumenten
- *.NEU.LOG mit allen außerhalb der spezifizierten Domäne befindlichen Dokumenten

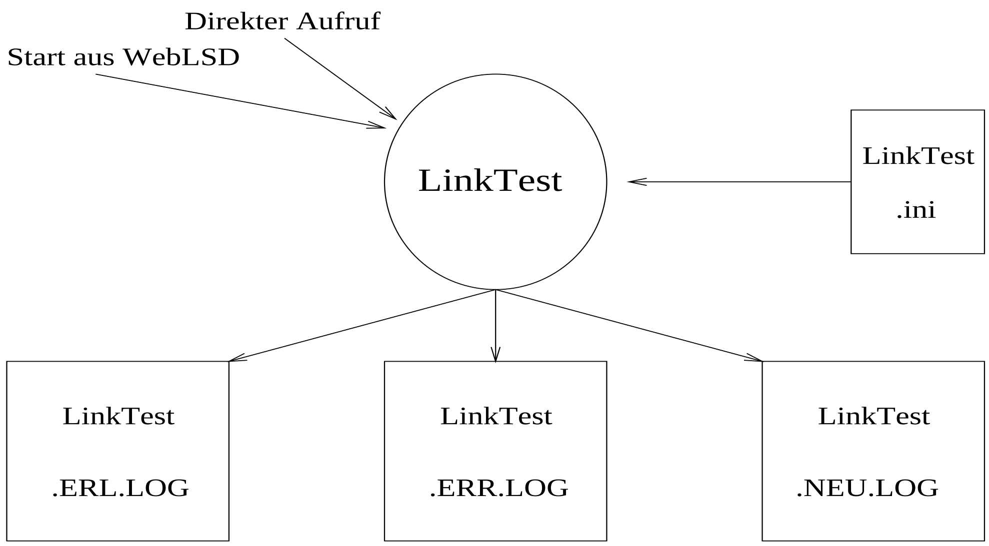

Abbildung 1.1: Das Programm LinkTest als Grundlage

Die Auswertung dieser Dateien ist die Basis für das entstehende Programm, sie enthalten alle zur Darstellung nötigen Daten wie Väter, Fehlerangaben oder URL des Dokuments und, falls eine Ermittlung möglich war, einen verantwortlichen Betreuer.

1.3 Aufgabenstellung

Implementierung eines Werkzeuges zur graphischen Darstellung von WWW-Link-Strukturen; so lautet die genaue Aufgabenstellung. Dahinter verbirgt sich das Problem, ausgehend von einer festzulegenden URL eine übersichtliche Darstellung eines solchen Geflechts zu realisieren. Im Normalfall handelt es sich dabei um nicht zyklenfreie, gerichtete Graphen.

Dabei ist jedoch offensichtlich, daß mit steigender Anzahl von HTML-Dokumenten eine geeignete Darstellungsform schwieriger zu finden ist. Eine Struktur kann immer so groß ausgewählt werden, daß eine adäquate Anzeige nicht mehr zu erreichen ist. Aus diesem Grund ist ein wesentlicher Bestandteil der Aufgabe eine Zoom-Funktion. Dies soll eine übersichtlichere Betrachtung von Teilen der Gesamtstruktur gestatten.

Aus der Sicht des Administrators sind nicht funktionierende Verweise die entscheidenden Punkte. Deshalb müssen solche fehlenden Dokumente auf den ersten Blick erkennbar sein. Bei einer graphischen Darstellung ist das durch verschiedene Farben recht einfach zu lösen. Sinnvoll ist dabei auch eine farbliche Abhebung von Verweisen, die auf einen anderen Server zeigen.

Zusätzlich müssen Informationen über die einzelnen Dokumente erhältlich sein. Angaben über die komplette URL sind hier genauso zu berücksichtigen, wie die Namen der Söhne und Väter, oder eines Betreuers. Bei einem fehlerhaften Link sollte eine Diagnose angeboten werden.

Als Grundlage dient das im letzten Abschnitt vorgestellte Werkzeug, das ebenfalls am Lehrstuhl entwickelt wurde. Vom Programm aus soll ein Aufruf dieses Werkzeugs, sowie das Auslesen der dabei entstehenden Logdateien möglich sein.

Zur Implementierung wurde die Programmiersprache Java vorgegeben, damit das Ergebnis möglichst plattformunabhängig bleibt und nicht zuletzt, um zu testen, ob die Sprache für diese Art von Aufgaben geeignet ist.

Kapitel 2

Die Benutzung von WebLSD

Für das entstandene Werkzeug haben wir den Namen WebLSD gewählt, als Acronym abgeleitet von **WebL**ink**S**tructure**D**isplay, da unsere Anwendung Linkstrukturen des WWW graphisch darstellt.

2.1 Aufbau des Fensters

In diesem Kapitel soll der Umgang mit dem Werkzeug näher erläutert werden. Zu Beginn sollen deshalb alle Komponenten kurz vorgestellt werden. In den darauffolgenden Abschnitten werden die einzelnen Punkte ausführlicher besprochen. Eine Momentaufnahme der Oberfläche zeigt Abb. 2.1

Das Fenster enthält eine Menüleiste mit den Punkten Datei, Bearbeiten und Hilfe. Im Datei - Menü sind die Kontrollelemente für das jeweilige Fenster enthalten, so zum Beispiel Schließen und Beenden. Das Bearbeiten - Menü beherbergt die Kontrolle über das Setup und die Ansteuerung des LinkTest Programmes. Hilfe wird für einen ersten Einstieg in WebLSD angeboten.

Eingerahmt von zwei Laufleisten, die es ermöglichen, auch größere Strukturen zu behandeln, befindet sich die Arbeitsfläche, auf der das Geflecht der Links dargestellt wird. Mit einem Mausklick auf ein beliebiges Dokument öffnet sich ein Popup, das unter anderem den Namen des Dokuments und die seiner Vorfahren oder auch eventuelle Fehler anzeigt.

Zwischen der Arbeitsfläche und der Menüleiste befindet sich noch eine Informationszeile, in der, in mehrere Angaben zerlegt, die URL des Dokuments angezeigt wird, über dem sich im Moment der Mauszeiger befindet.

2.2 Menüstruktur

2.2.1 Das Datei - Menü

Die erste Überschrift in der Menüleiste trägt den Namen **Datei**. Nach deren Auswahl öffnen sich die folgenden vier Menüpunkte:

- **Aktualisieren** veranlaßt einen Neustart des aktuellen Fensters mit dem aktuellen Setup. Dieser Punkt findet seine Anwendung nach einem Durchlauf des Link Test Programms oder nach veränderter Auswahl von LOG-Dateien im Setup.

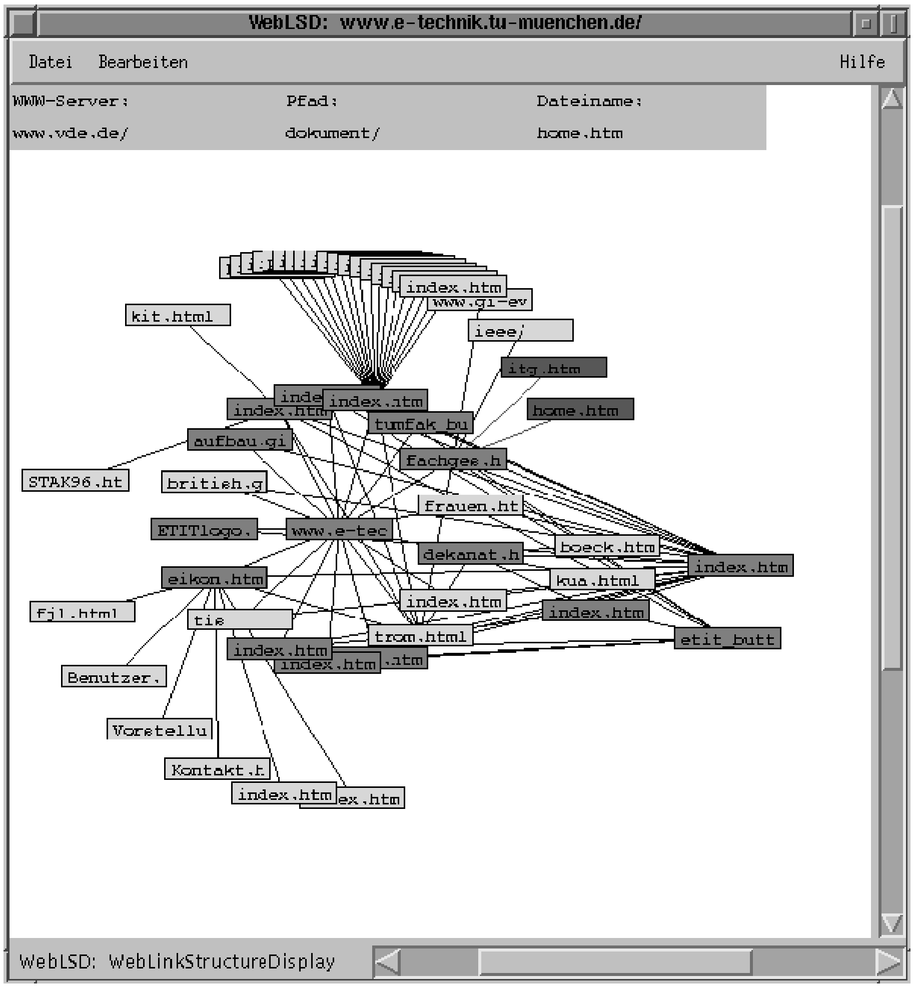

Abbildung 2.1: WebLSD nach dem Start

- **Neues Fenster** erstellt eine Kopie des aktuellen Fensters. Damit läßt sich ein einfacher Vergleich zwischen zwei Läufen des Link Tests ermöglichen. Zu diesem Zweck wird nach dem Beenden des neuen Tests Datei - Neues Fenster ausgewählt und in diesem Datei - Aktualisieren durchgeführt. Die beiden Inkarnationen von WebLSD enthalten nun wie gewünscht die Zustände vor und nach dem Link Test.
- **Schließen** beendet das aktuelle Fenster. Alle anderen Fenster bleiben davon unberührt. Wenn nur ein Fenster geöffnet ist, ist der Befehl mit Beenden identisch.
- **Beenden** beendet den gesamten Programmlauf. Alle anderen Fenster werden ebenfalls geschlossen.

2.2.2 Das Bearbeiten - Menü

Unter der Überschrift Bearbeiten befindet sich die Verwaltung und Bearbeitung der Setup-Datei und eine Startmöglichkeit für das Link Test Programm. Konkret finden sich diese drei

Menüpunkte:

- **Setup laden** öffnet ein Fenster mit dem eine abgespeicherte Initialisierungsdatei ausgewählt werden kann. Ist die Datei korrekt, so startet anschließend das Fenster zum Bearbeiten des Setup (siehe nächster Punkt). Ist sie nicht im richtigen Format (z.B. falsche Datei ausgewählt), dann wird die Aktion ignoriert, es erscheint eine Fehlermeldung auf der Standardausgabe.

- **Setup bearbeiten** öffnet, wie der Name vermuten läßt, ein Fenster mit dem das aktuelle Setup bearbeitet werden kann. Im Setup müssen folgende Parameter konfiguriert werden:

  - Der absolute (d.h. mit kompletter Pfadangabe) Name des Programmes, das den Link Test durchführt;
  - der absolute Name der LOG-Dateien, ohne die Namenserweiterung (d.h. nur der gemeinsame Teil der Namen ohne .XXX.LOG);
  - die Auswahl der anzuzeigenden Dokumente (alle, nur die fehlerfreien oder nur die getesteten);
  - die komplette Namensangabe eines verwendbaren WWW-Browsers an, eine leere Eingabe kennzeichnet, daß kein Browser verwendet werden soll;
  - wenn die Datei abgespeichert werden soll, muß zusätzlich noch der gewünschte Name angegeben werden.

  Das Fenster läßt sich über einen von drei Knöpfen schließen. **OK** behält die Änderungen, speichert die Datei aber nicht ab, **Speichern** behält die Veränderungen ebenfalls, speichert sie jedoch zusätzlich unter dem vorgegebenen Namen ab und **Cancel** verwirft die Änderungen.

- **Struktur testen** startet im Hintergrund einen Prozess, der das im Setup spezifizierte Link Test Programm ausführt. Nach Beendigung wird je nach Erfolgs- oder Fehlerfall eine entsprechende Meldung eingeblendet.

2.2.3 Das Hilfe - Menü

Das Hilfe - Menü befindet sich rechts von den beiden anderen und weist zwei Punkte auf:

- **Über** blendet eine Meldung ein, auf der die Namen der Autoren sowie die Technische Universität und das Institut für Informatik vermerkt sind.

- **Informationen** gibt einige kurze Tips zum Direkteinstieg ohne schriftliche Dokumentation.

2.3 Arbeitsfläche

Auf der Arbeitsfläche wird die Linkstruktur dargestellt. Im Zentrum befindet sich die Start-URL. Zyklisch darum angeordnet sind die Dokumente, auf die die Links verzweigen. Jeder neue Kreis stellt eine weitere Generation von Verzweigungen dar.

Fehlerhafte beziehungsweise fehlende Dokumente sind durch rote Farbe dargestellt. Verzweigungen auf fremde Server sind in gelb gehalten, während fehlerfreie Dokumente des getesteten

Servers in grün abgebildet sind.

Wenn man mit dem Mauszeiger Dokumente berührt, wird in einer Informationszeile die URL dargestellt (genaueres dazu im nächsten Abschnitt).

Bei einem Klick auf dieses Dokument wird ein zugehöriges Popup-Fenster angezeigt, das wiederum mit Mausklick geöffnet wird. An oberster Stelle steht der **Name** des Dokuments. Falls auf dieses nicht zugegriffen werden konnte, steht darunter die **Fehlerursache** knapp erläutert. Für exaktere Fehlerbeschreibungen muß auf das LinkTest Paket zurückgegriffen werden. Wenn von diesem Dokument noch weitere Verzweigungen ausgehen, so wird der Punkt **Zoom auf Unterstruktur** angeboten, der eine neue Inkarnation von WebLSD startet, dessen Ausgangsdokument das eben ausgewählte ist. Im Anschluß daran sind alle **Verweise auf dieses Dokument** aufgeführt, die von getesteten Dokumenten stammen. Die Option **Dokument anzeigen** startet in einem eigenen Prozess den im Setup spezifizierten WWW-Browser mit der ausgewählten Seite. Der letzte Punkt dieses Popups lautet **Popup schließen** und macht genau das.

2.4 Informationszeile

Die vorher bereits angesprochene Informationszeile zeigt also, wie oben erwähnt, die URL des Dokuments, über das sich im Moment der Mauszeiger bewegt. Dabei wird die URL in drei Komponenten zerlegt:

Links steht der Name des Servers, auf dem sich das Dokument befindet. In der Mitte ist der Pfad des Dokuments auf diesem Server aufgeführt und rechts steht der Name des Dokuments selbst. Dabei ist allerdings zu beachten, daß ein sehr langer Name nicht vollständig dargestellt wird, sondern nur jeweils die ersten 24 Zeichen, was aber in den meisten Fällen ausreichend ist. Nur sehr tief verschachtelte Pfade oder lange Subnetznamen sprengen diesen Rahmen.

Kapitel 3

Der Entwurf von WebLSD

3.1 Realisierung in JAVA

Da die Aufgabenstellung die Plattformunabhängigkeit der Anwendung verlangt, sollte dafür Java als Programmiersprache verwendet werden. Java erreicht diese Plattformunabhängigkeit durch Interpretation, ohne auf die Vorteile einer Compilersprache zu verzichten.

Die Programmierumgebung von Java beinhaltet sowohl einen Compiler, als auch einen Interpreter. Der Sourcecode, der in einer `.java` - Datei abgespeichert ist, wird mit dem Aufruf `javac file.java` vom Compiler in einen Binärcode für eine virtuelle Maschine umgewandelt und in einer `.class` - Datei abgespeichert. Das compilierte Programm wird mit dem Befehl `java file` aufgerufen. Dabei wird dieser Binärcode zur Laufzeit durch einen Interpreter, der die virtuelle Maschine realisiert, in maschinenspezifische Befehle umgewandelt, ganz im Gegenteil zu anderen Programmiersprachen wie C oder C++, wo beim Compilieren der Sourcecode direkt in Maschinencode übersetzt wird, der dann nur auf den jeweiligen Zielsystemen ausgeführt werden kann (siehe Abb. 3.1).

Dieser Interpretationssschritt erfordert entgegen jeder Vermutung allerdings nicht erheblich mehr Zeit, da die Umwandlung des Binärcodes in tatsächliche Maschinenbefehle mit wenig Aufwand und Rechnerleistung möglich ist. Die Ausführungsgeschwindigkeit wird dadurch nicht so deutlich beeinträchtigt, wie es bei anderen ausschließlich interpretierten Sprachen der Fall ist.

Also ist ein Interpreter unbedingt erforderlich, um Java-Programme ausführen zu können. Deshalb erfordert jeder Java-fähige Browser, wie z.B. die Version 2.x des Netscape Navigators oder der Sun-eigene HotJava Browser, die Einbindung eines Interpreters. Dieser ermöglicht dann den Ablauf des Programms auf dem abrufenden Rechner.

Die Entwicklungsumgebung von Java, das sogenannte Java Developer‘s Kit (JDK), steht für unterschiedliche Systeme, wie z.B. Sun Solaris 2.3-2.5 oder Windows95, zur Verfügung, wobei die virtuelle Maschine schon für mehr Systeme verfügbar ist. So kann auf nahezu jeder Umgebung, die Threads und lange Dateinamen zur Verfügung stellt, mit Java gearbeitet werden.

Außerdem sollte auch untersucht werden, mit welchem Aufwand ein Graphical User Interface (GUI) in Java erstellt werden kann und welche Möglichkeiten diese Programmiersprache hierfür zur Verfügung stellt.

Java-Anwendungen können als Applications oder Applets erstellt werden. Applets sind in Java erstellte und in Web-Seiten eingebundene Programme, die über das WWW heruntergeladen und von einem Browser auf dem Rechner des Benutzers ausgeführt werden. Dahingegen sind Applications selbständige Programme, so wie sie auch mit jeder anderen Programmiersprache

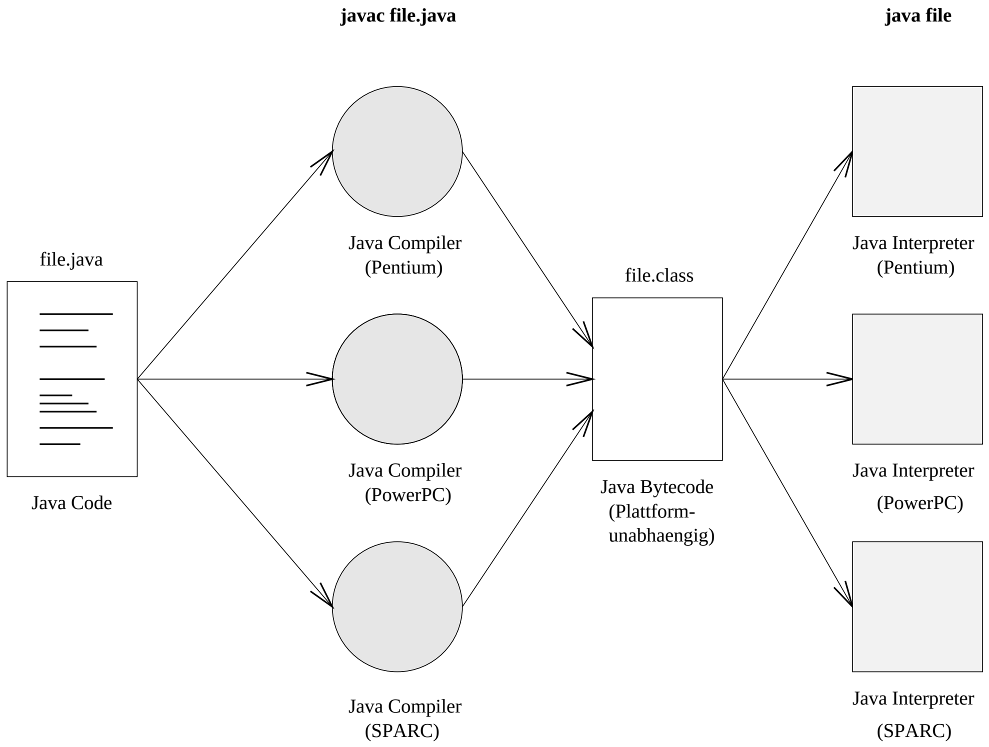

Abbildung 3.1: Erstellung von Java-Programmen

erstellt werden können. Sie erfordern keinen Browser, um ausgeführt zu werden. Zusätzlich haben Applications im Vergleich zu Applets die Möglichkeit, auf das Dateisystem eines Rechners zuzugreifen, während das bei den Applets aus Sicherheitsgründen unterbunden wurde. So soll verhindert werden, daß Applets Schäden auf dem abrufenden Rechner anrichten können.

Es gibt allerdings auch Programme, die sowohl als Applet als auch als Application ausführbar sind. Wir haben unser Programm als eigenständige Applikation implementiert, da unbedingt Zugriff auf das Dateisystem notwendig ist, um benötigte Informationen auslesen zu können. Ohne diesen Zugriff wäre diese Anwendung nicht realisierbar gewesen.

3.2 Die Grundstruktur von WebLSD

Das Programm gliedert sich logisch in vier Stufen (siehe dazu Abb. 3.2), die hier zur Übersicht kurz und in den nachfolgenden Kapiteln ausführlicher beschrieben sind.

Das Programm soll, auf ein anderes Werkzeug zur Konsistenzprüfung aufbauend, die Verbindungsstrukturen zwischen WWW-Dokumenten darstellen. Dazu ist es als erstes nötig, die von diesem Werkzeug zur Verfügung gestellten LOG-Dateien auszuwerten. Dabei werden die Informationen über URL, Väter, sowie gegebenenfalls Betreuer und Fehlerangaben im ersten Schritt eingelesen und gespeichert.

Im nächsten Schritt wird die gesamte Verbindungsstruktur erzeugt, in diesem Falle ein einfaches Feld, in dem die Verknüpfungen zu den Vorfahren und Nachkommen über die Feldindizes nachgebildet werden. Die dabei entstehende Datenstruktur ist die Grundlage für alle weiteren

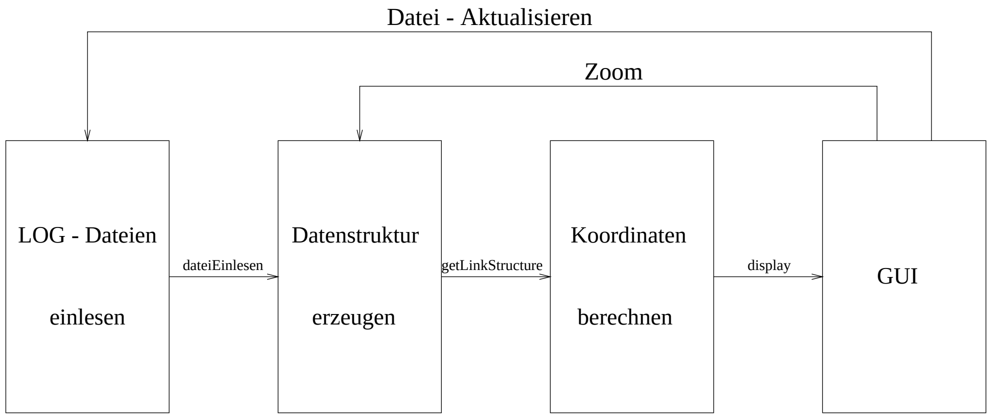

Abbildung 3.2: Vier Schritte zur Darstellung

Schritte.

Diese Datenstruktur durchläuft anschließend einen rekursiven Algorithmus, in dem die Darstellungsorte der einzelnen Dokumente so berechnet werden, daß sie sich in eine Gesamtdarstellung aller Verbindungen einfügen. Das Verfahren arbeitet dabei wie eine Breitensuche, das heißt, es werden alle Nachkommen einer Generation bearbeitet, bevor der erste Nachkomme einer weiteren Generation behandelt wird.

Der letzte Schritt ist die eigentliche Darstellung der Linkstruktur in einer graphischen Benutzeroberfläche (GUI = Graphical User Interface). Hier kann der Anwender neben der Gesamtansicht auch alle weiteren Informationen ermitteln. Von dieser GUI aus ist es ebenfalls möglich, nach einer erneuten Konstistenzprüfung vom ersten Schritt ab neu zu beginnen (Display aktualisieren), oder auf dem zweiten Schritt aufsetzend eine Teilstruktur genauer zu betrachten (Zoom).

Die ersten drei Schritte sind im Package `documentDataStructures` implementiert, das aus mehreren aufeinander aufbauenden Klassen (`BasisDokument, VollDokument, Dokument,` sowie `LinkStructure`) besteht. Hier sind auch alle zur Arbeit mit der Datenstruktur benötigten Funktionen bereitgestellt. Relativ zur Basisklasse mit dem Hauptprogramm muß ein solches selbsterstelltes Package in einem Unterverzeichnis mit dem Namen des Packages stehen. Es existiert in diesem Verzeichnis je eine Datei mit dem Quelltext und eine mit compiliertem Bytecode für jede Klasse. Die GUI selbst, mit dem Hauptprogramm und allen benötigten Unterklassen, ist in der Datei `WebLSD.java` definiert. Hier existiert nach der Übersetzung zu jeder intern definierten Klasse ebenfalls eine Datei mit Bytecode.

Damit das Werkzeug möglichst flexibel bleibt, werden die Grundeinstellungen in der Initialisierungsdatei `WebLSD.ini` abgespeichert. Sie besteht aus der Angabe über die darzustellenden Dokumente (alle, fehlerfreie und getestete Dokumente), den absoluten Namen des Link - Test - Programms, der Log - Dateien und eines installierten WWW-Browsers. Da diese Namen die einzigen vom Betriebssystem bzw. der Plattform bestimmten Faktoren sind, wird mit einer freien Konfigurierbarkeit die Unabhängigkeit davon sichergestellt. Die Setup - Datei wird beim Programmstart ausgewertet und beeinflußt dementsprechend den weiteren Ablauf. Bei Nichtexistenz oder fehlenden Angaben werden Standardwerte verwendet, die eine fehlerfreie Ausführung aber nicht immer sicherstellen. So sind zum Beispiel Pfadangaben für eine UNIX-Plattform unter

Windows syntaktisch nicht korrekt und umgekehrt.

3.3 Einlesen der LOG-Dateien

Wie bereits erwähnt, stellt das Basiswerkzeug als Schnittstelle drei ASCII - Dateien mit korrekten, fehlerhaften und nicht getesteten Dokumenten zur Verfügung, die ausgewertet werden müssen. Ein Datensatz ist dabei wie folgt aufgebaut:

```
Url       : HTTP://www.vde.de/dokument/fachges/itg.htm
Vater     : www.e-technik.tu-muenchen.de/fachges.html:Z21
Fehler    : 404
Intern --: www.vde.de/dokument/fachges/itg.htm;HTTP;
           www.e-technik.tu-muenchen.de/fachges.html:Z21;404;A
```

Die Zeile mit `Intern --:` beinhaltet alle Daten, die übrigen Zeilen dienen nur der besseren Lesbarkeit. Zur Auswertung existiert eine Methode `void DateiEinlesen(String dateiName)`, die für jede der LOG - Dateien einzeln aufgerufen werden muß. Dies ermöglicht über den Setup-Parameter `logFiles` die Auswahl der darzustellenden Dokumente, indem die Datei nicht gewünschter Elemente einfach nicht ausgewertet wird.

Die Prozedur öffnet die angegebene Datei und liest in einer Schleife jeweils eine Zeile in einen String. Enthält dieser obige Markierung, wird er durch die Funktion `getEntry( String input)`, die diesen String nach den Feldtrennern durchsucht, in folgende Bestandteile zerlegt:

- URL des Dokuments
- Zugriffsmethode auf das Dokument
- Array mit den URLs aller Väter
- Fehlernummer oder Betreuer wenn vorhanden

Jeder relevante Datensatz (z.B. mailto's sind zur Darstellung vernachlässigbar), wird dem Vektor `gefundeneDokumente` angefügt. Ein Vektor ist im wesentlichen ein beliebig langes, mit verschiedenen Datentypen belegbares Feld; er wird standardmäßig in java.util zur Verfügung gestellt.

Bei fehlerhafter oder fehlender Datei wird eine Meldung auf die Standardausgabe geschrieben, und die Prozedur beendet. Der Vektor enthält im Fehlerfalle alle Dokumente bis zu dem, das den Fehler verursacht hat, bei fehlender Datei logischerweise keine weiteren Informationen.

Nach dem erfolgreichem Durchlauf der Prozedur mit allen ausgewählten LOG - Dateien, enthält der Vektor schließlich alle Datensätze, wie sie vom Link - Test übergeben wurden.

3.4 Erzeugung der Datenstruktur

Aus dem Vektor mit den ermittelten Datensätzen wird anschließend die vom Programm benutzte Datenstruktur erzeugt (siehe dazu Abb. 3.4).

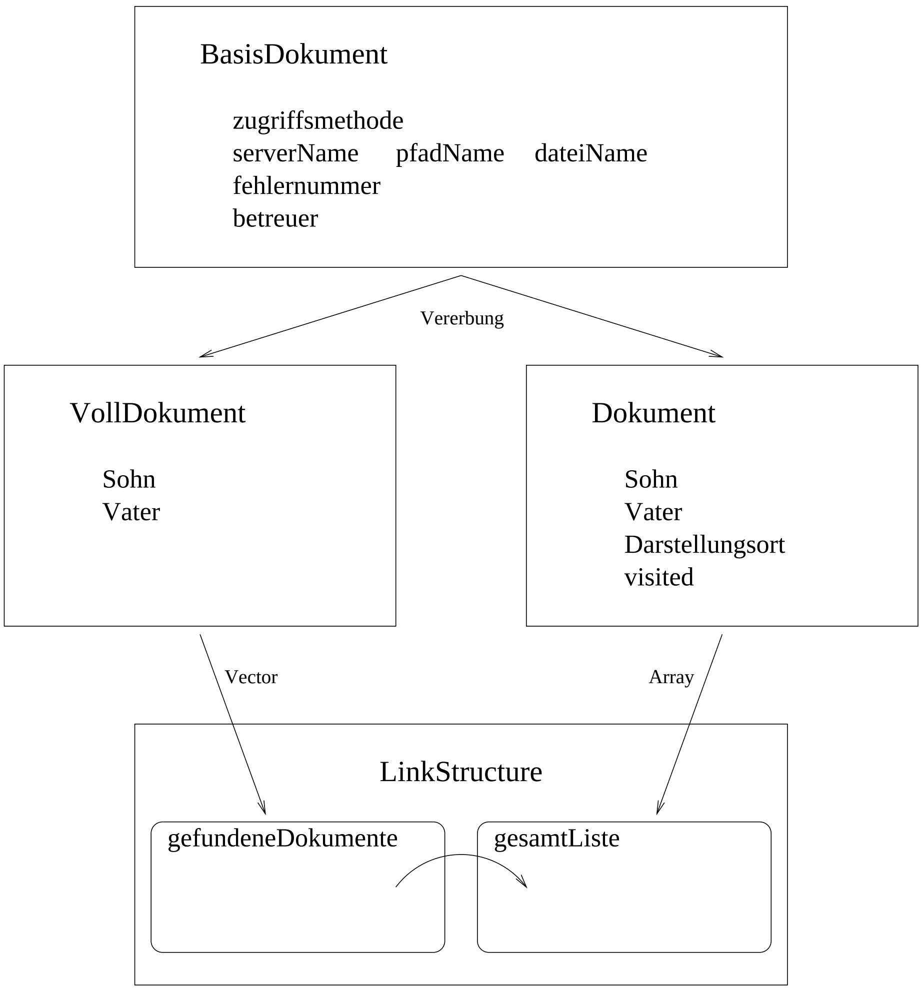

Abbildung 3.3: schematischer Aufbau der Datenstruktur

Diese besteht aus dem in der Klasse `LinkStructure` definierten Feld `gesamtListe[]` von Dokumenten. Jedes Dokument besteht dabei grundsätzlich aus der URL, die aufgespalten ist in `zugriffsmethode`, `serverName`, `pfadName` und `dateiName`, sowie einer eventuell vorhandenen `fehlernummer` oder einem `betreuer`. Diese Elemente sind in der Klasse `BasisDokument` festgelegt. In der darauf aufbauenden Klasse `Dokument` ist noch eine zusätzliche Variable `visited` für eine Kennzeichnungsmöglichkeit bei Bearbeitungsalgorithmen eingefügt. Außerdem existieren hier zwei Felder `sohn[]` und `vater[]`, mit deren Hilfe die Verküpfungen realisiert werden. Der Inhalt dieser Felder sind lediglich die Indizes der Nachkommen und Vorfahren des aktuellen Dokuments.

In der Funktion `getLinkStructure(Vector gefundeneDokumente)` werden die Informationen der einzelnen Datensätze in die Liste der Dokumente übernommen. Da in den LOG-Dateien lediglich die Väter verzeichnet sind, muß die Struktur in umgekehrter Richtung erst aufgebaut werden. Dies geschieht in zwei verschachtelten Schleifen, in denen überprüft wird, ob eine URL als Vater einer anderen auftaucht. Ist dies der Fall, werden deren Indizes in `vater[]` und `sohn[]`

der entstehenden Datenstruktur entsprechend gesetzt. Beim Startdokument der Link-Struktur ist als Vater der String TOP eingetragen, dementsprechend kann die Variable `startDokument` belegt werden.

Nach dem Durchlauf dieser Funktion wird der Vektor `gefundeneDokumente` nicht mehr benötigt, er kann an dieser Stelle freigegeben werden.

3.5 Berechnung der HTML Struktur

Zur graphischen Umsetzung der Verbindungsstruktur gäbe es eine Vielzahl von Möglichkeiten. Eine zu Beginn angestrebte Darstellung als Baum hat sich bei näherer Betrachtung als ungeeignet erwiesen, da sich der so entstehende Baum sehr in die Breite entwickelt und damit schnell unübersichtlich wird. Außerdem verleitet ein baumartiges Bild dazu, den zyklischen Graphen, der ja eigentlich dahintersteckt, zu übersehen.

Wir haben uns deshalb entschieden die Dokumente kreisförmig um das Startdokument anzuordnen, das so die Koordinaten (0,0) erhält. Die rekursive Funktion `Display(int from, int to, LinkStructure struktur, int dokument)`, die die restlichen Koordinaten berechnet (siehe dazu Bild 3.5), arbeitet als Breitensuche. Das heißt, alle noch nicht behandelten Söhne des aktuellen Dokuments werden gleichmäßig auf einem Kreisbogen mit dem um die Konstante `addRadius` vergrößertem Radius angeordnet, bevor die Rekursion fortschreitet. Jedem Dokument wird dabei der Winkelbereich zugeordnet, der sich aus dem Winkelbereich seines Vaters, im Falle des Startdokuments 0 bis 360 Grad, dividiert durch die Anzahl dessen unbesuchter Söhne ergibt. Dieser Winkelbereich steht nun wiederum diesem Dokument zur Verfügung, um seine Nachkommen auf dem nächstgrößerem Kreis zu plazieren. Aus diesem Winkelbereich, dem Radius sowie der Anzahl der unbesuchten Söhne lassen sich nun die jeweiligen Koordinaten bestimmen.

Da manche HTML-Seiten sehr viele Verzweigungen aufweisen (z.B. eine Indexdatei eines Servers oder eine Seite mit Bookmarks), wird es in solchen Fällen oft zu sehr dicht angehäuften Darstellungen kommen, die nur schwierig erkennbar sein dürften. Andere Seiten hingegen (vor allem Bilder o.ä.) besitzen überhaupt keine weiteren Verknüpfungen, oder nur solche, die bereits an anderer Stelle wiedergegeben wurden.

Um die Anhäufungen etwas zu entzerren, liefert die Rekursion die Anzahl der im Kreisbogen untergebrachten Nachkommen zurück. Ist dieser Wert gleich null, d.h. der Sektor verbleibt ohne Nachkommen, wird dieser Winkelbereich dem nachfolgendenfolgenden Dokument zur Verfügung gestellt. Sollten sich diese Winkel auf mehr als 90 Grad aufsummieren, wird der Bereich auf diesen Wert gekürzt, denn eine zu große Streuung bei weit vom Mittelpunkt entfernten Dokumenten hat sich als unvorteilhaft erwiesen.

Da das Koordinatensystem bei Graphiken in Java den Ursprung in der linken oberen Ecke des Fensters hat und nur positive Werte erlaubt, müssen im Anschluß an die Berechnungen die Koordinaten entsprechend berichtigt werden. Dazu wird der größte benutzte Radius während der Berechnung in der Variable `maxRadius` festgehalten und dieser Wert bei allen Dokumenten in x- und y- Richtung aufaddiert. Die so korrigierten Daten können jetzt zur weiteren Verwendung direkt benutzt werden.

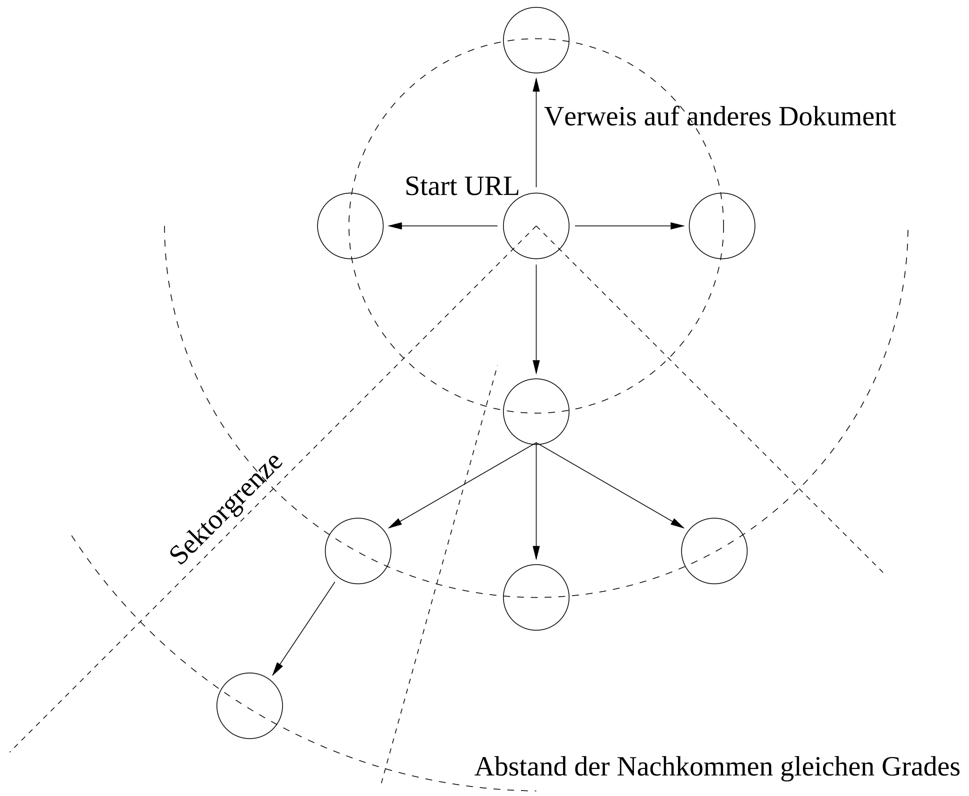

Abbildung 3.4: Berechnung der Koordinaten

3.6 Die graphische Benutzeroberfläche

3.6.1 Aufteilung des Fensters

Eine genaue Erklärung zur Implementierung aller Bestandteile des Fensters erscheint an dieser Stelle überflüssig, da sie größtenteils durch die Programmiersprache vorgegeben und in den Quelltexten des Anhanges leicht verständlich vorzufinden sind. Im folgenden ist lediglich die grundsätzliche Aufteilung, unter Nennung der verwendeten Klassen, erläutert.

Für die Oberflächengestaltung in Java steht das Abstract Windowing Toolkit `java.lang.awt` mit seinen Klassen zur Verfügung. Das Fenster, erzeugt durch die Klasse `WebLSD`, die von `Frame` abstammt, teilt sich in zwei Bereiche auf:

- Zum einen die Menüleiste mit den Optionen Datei - Bearbeiten - Hilfe, die mit der Klasse `MenuBar` realisiert ist. Die Auswertung eines solchen Menüs erfolgt durch Überschreiben der vorgegebenen Methode `handleEvent(Event evt)`, die bei jedem Ereignis aufgerufen wird und eben dieses Ereignis übergibt. Mit der Variable `Event.target` wird es dort in einer `switch` Anweisung dem jeweiligen Menüpunkt zugeordnet.

- zum anderen ist durch die Auswahl des `BorderLayout` schließlich der Rest des Fensters vorgegenben. Diese Layoutvariante teilt eine Fläche in fünf Teile auf: `Center, North, East, South` und `West`. Davon sind, wie der Name schon andeutet letztere rundum am

Rand angeordnet, ersteres erhält den restlichen verfügbaren Platz. In unserem Falle sind als Umrandung nur rechts und unten zwei Laufleisten durch die Klasse `Scrollbar`, sowie ein Label mit dem Namen des Programms eingefügt. Im `Center` befindet sich die Arbeitsfläche, die in der gleichnamigen Klasse implementiert ist. Diese wiederum ist als `PositionLayout` ausgeführt, was freie Positionierung ermöglicht. Die `Arbeitsfläche` setzt sich zusammen aus einer Informationszeile, der im nachfolgenden Kapitel erläuterten Darstellung und einem PopUp, welches bei Mausklick zu dem Dokument eingeblendet wird, über dem sich der Mauszeiger gerade befindet. Dieses `PopUp` besteht lediglich aus einer Erweiterung der Klasse `Choice`. In der Informationszeile wird durch `Labels` - in drei Teile zerlegt - die URL des im Moment mit der Maus berührten Dokuments angezeigt. Die drei Bestandteile sind der Name des Servers, die Pfadangabe auf diesem Server und der Dateiname.

3.6.2 Darstellung der HTML - Struktur

Das Zeichnen der ermittelten Struktur erfolgt in der Klasse `Arbeitsfläche` durch die Prozedur `paint(Graphics g)`, die vom System aufgerufen wird, wann immer eine Darstellung des betreffenden Fensterbereichs notwendig wird, oder wenn es über `repaint()` explizit angefordert wird, beispielsweise nach einem Scrollereignis.

Um den Stand der beiden Scrollbars bei der Anzeige korrekt wiederzugeben, existiert in der `Graphics`-Klasse die Prozedur `translate(int xWert, int yWert)`, die, abhängig von den übergebenen Werten, den aktuellen Ausschnitt der Grafik auswählt. Diese Prozedur wird beim Eintritt in die `paint`-Methode als erstes aufgerufen. Die zugehörigen Werte werden bei allen Scrollereignissen entsprechend verändert, und in den Variablen `tx` und `ty` der Arbeitsfläche festgehalten.

Im Anschluß werden durch die Prozedur `paintEdges(Graphics g)` alle Links dargestellt. Dazu wird eine Schleife durchlaufen, die von der Koordinate jedes Dokuments mittels der Methode `edges(Graphics g, int dokument)` die Verbindungslinien zu den Koordinaten aller Söhnen dieses Dokuments zieht. Bei fehlerhaften Verbindungen wird die Linie rot, ansonsten schwarz gezeichnet.

Dann werden durch den Aufruf von `paintNodes(Graphics g)` die Dokumente darübergelegt. Auch hier wird nur eine Schleife durchlaufen, in deren Verlauf um die Koordinaten aller Dokumente ein Kästchen mit Rahmen gezeichnet wird. Im Fehlerfalle wird dessen Hintergrund rot, bei Dokumenten auf fremden Servern gelb, ansonsten grün eingefärbt. In dieses Kästchen wird der Name des Dokuments geschrieben, soweit durch die beschränkte Länge machbar. Im vorliegenden Programm sind dies neun Zeichen des Dateinamens, beziehungsweise der Pfad- oder Serverbezeichnung, falls kein Dateiname vorhanden ist.

Kapitel 4

Zusammenfassung

4.1 Voraussetzungen

Um WebLSD verwenden zu können, sind nur wenige Voraussetzungen notwendig. Das einzig benötigte ist ein Interpreter für den Java - Bytecode der virtuellen Maschine mit den zugehörigen Standardklassen. Dieser ist beispielsweise im Java-Development-Kit enthalten, das inzwischen bereits auf nahezu alle Plattformen portiert wurde, die lange Dateinamen und Parallelität unterstützen (Sun Solaris, HP - UX, AIX, Linux, Windows95, Windows NT, OS/2, Mac OS und andere). Es existieren aber auch andere Entwicklungs- und Laufzeitumgebungen (z.B. von Symantec oder Borland), die dieselbe Funktionalität bieten, allerdings keine frei verfügbare Software sind. Selbst für Systeme, die obige Grundvoraussetzungen nicht erfüllen (DOS, Windows 3.x) entstehen derzeit Portierungen. Damit sollte also ausreichende Hardwareunabhängigkeit gegeben sein.

Ansonsten muß lediglich ein Verzeichnis angelegt werden, in dem sich alle `.class`-Dateien von WebLSD befinden und ein Unterverzeichnis mit dem selben Namen wie das Package **`documentDataStructures`**, samt den zugehörigen `.class`-Dateien. Damit ist das Programm bereits lauffähig.

Es ist ebenso möglich, das Package in ein Standardverzeichnis für Java-Klassen zu kopieren. Die Umgebungsvariable `CLASSPATH` muß dazu entsprechend gesetzt werden. Da diese Definitionen aber nicht gerade zu Standardbibliotheken gehören, sondern nur speziell für dieses Werkzeug entwickelt wurden ist diese Vorgehensweise nicht unbedingt zu empfehlen.

Für einen sinvollen Betrieb ist natülich auch das Werkzeug zum Testen der HTML-Struktur [Sch96] notwendig, das unter PERL abläuft, denn diese Funktionalität erbringt WebLSD, wie erläutert, nicht selbst.

Da bei einer Neuinstallation vermutlich die Konfigurationsdatei nicht mehr verwendet werden kann, empfiehlt es sich, diese vor dem ersten Start neu zu erstellen, da zum Beispiel bei einem Plattformwechsel die verwendeten Pfadangaben nicht einmal mehr syntaktisch korrekt sein müssen und dadurch sogar zum Absturz des Werkzeuges führen können.

Die Angaben in der Datei bestehen aus ASCII-Zeichen und können mit jedem beliebigen Texteditor erstellt werden. An erster Stelle stehen die anzuzeigenden Dokumente (0 = alle; 1 = fehlerfreie; 2 = getestete Dokumente). In den nächsten Zeilen stehen der Reihe nach die vollständigen Namen der LOG-Files (ohne Erweiterungen .XXX.LOG), des LinkTest Programms, sowie des zu verwendenden WWW-Browsers, falls ein solcher aus WebLSD heraus benutzbar sein soll.

4.2 Einschränkungen und Probleme

Einige kleinere Mängel der Anwendung ergeben sich aus der Programmiersprache. So ist es in Java nicht möglich, die Koordinaten des Fensters auf dem Gesamtbildschirm zu ermitteln, es kann auch nicht vom Programm aus frei positioniert werden und öffnet sich deshalb immer an der gleichen Stelle. Wenn also nach Auswahl des Menüpunkts Datei - Aktualisieren die Struktur neu eingelesen und dabei das augenblickliche Fenster geschlossen wird, öffnet sich das neue Fenster mit der aktualisierten Darstellung nicht an der vorherigen Stelle, falls das alte vorher verschoben wurde.

In der Informationszeile mit der in drei Teile zerlegten Anzeige der URL werden längere Namen (z.B.: www.nm.informatik.uni-muenchen.de) aus Platzmangel nicht vollständig wiedergegeben. Eine flexible Breite ist ebenfalls nicht möglich, da die Gesamtbreite des Fensters beschränkt ist, alle Angaben zusammen aber beliebig lange werden können. Eine Verteilung auf mehrere Zeilen war durch eine Einschränkung des verwendeten Java-Toolkits nicht machbar: Die Aufteilung des Feldes erfolgt mit der Definition `setLayout( new GridLayout(6,2))` auf sechs Einheiten in der Breite mal zwei in der Höhe. Ein Festlegung `setLayout( new GridLayout(2,6))` führte allerdings aus unbekannten Gründen zu keinerlei Veränderungen.

Die gewählte Darstellungsweise führt außerdem dazu, daß sehr große Server und Dokumente mit sehr vielen Verweisen nicht sehr übersichtlich zu betrachten sind.

Dazu gibt es zwar die Zoomfunktion, aber bei einer zu hohen Anzahl von Links, die vom Startdokument abgehen, steht es auch hier mit der Übersicht nicht zum besten. Für solch außergewöhnlich große Strukturen, müssten geeignetere Administrationswerkzeuge entwickelt werden, die in der Darstellung flexibler sind.

4.3 Abschlußbemerkung

Als Fazit läßt sich festhalten: Java ist durchaus geeignet, um Aufgaben dieser Art zu lösen. Alle gestellten Anforderungen wurden gelöst und das entstandene Werkzeug ist voll funktionsfähig. Erweiterungsmöglichkeiten wie eine bessere Strukturierung des Aufbaus für sehr komplexe Strukturen oder gar eine freie Konfigurierbarkeit der Darstellung, wären als Erweiterungen ebenfalls möglich, konnten aber nicht im Rahmen dieses Fortgeschrittenenpraktikums realisiert werden.

Allgemein sind in Java graphische Oberflächen recht einfach zu erstellen, die dazu mitgelieferten Klassen und Methoden genügen den meisten Anforderungen. Sehr rechenintensive Anwendungen sollten allerdings nicht in Java implementiert werden, was aber eine Einschränkung für alle interpretierten Sprachen ist. Zu diesem Zweck ist jedoch die Einbindung von C- Programmen problemlos möglich, die Plattformunabhängigkeit ist dadurch logischerweise verloren. Wenn die Kinderkrankheiten von Java, die sich aber vor allem auf Aspekte der Anwendungen als Applet beschränken, wie von Sun angekündigt in Bälde beseitigt sind, steht eine einfache aber mächtige Sprache zur Verfügung, die noch einiges Potential für Anwendungen in Netzwerken und verteilten Umgebungen in sich birgt.

Anhang A

JAVA - ein kurzer Überblick

JAVA !!! Jeder schwärmt von dieser neuen Programmiersprache, doch nicht jeder weiß genau, was sie für Möglichkeiten bereitstellt. Java wurde von der Firma Sun entwickelt und wird meistens zur besseren Aufbereitung von WWW-Seiten verwendet, da man durch Einbinden von Ton und bewegten Bildern multimediale Anwendungen darstellen kann. In Verbindung mit der gestiegenen Popularität des Internet ist dies sicherlich ein Faktor, der zur weiten Verbreitung von Java beigetragen hat. Java soll die Entwicklung von sicheren und robusten Anwendungen ermöglichen, die keine Fehler- oder Gefahrenquelle für ein gesamtes Netz oder auch nur für einen einzelnen Rechner darstellen sollen.

A.1 Eigenschaften von Java

Doch Java kann mehr. Die Absicht der Entwickler war, eine sehr leicht erlernbare Programmiersprache zu entwickeln, die sich an C++ hält und daraus nur die unnötigen Komplexitäten ausklammert.

Da Java objektorientiert gestaltet ist, werden neue Methoden der Softwareentwicklung ebenso unterstützt, wie verteilte Systemumgebungen von Client-Server-Strukturen. Außerdem soll man sich aus Sicherheitsgründen darauf verlassen können, daß die Anwendungsprogramme stabil und sicher laufen, da Java hauptsächlich für verteilte Systeme entwickelt wurde. Mit Hilfe einer Laufzeit-Speicherverwaltung werden die Anwendungsprogramme stabil gestaltet. Pointer sowie Pointerarithmetik sind in Java nicht erlaubt. Sowohl während des Compilierens, als auch zur der Laufzeit, werden Sicherheitsüberprüfungen durchgeführt. Zusätzlich sind Schutzmechanismen gegen einen möglichen Zugriff von außen eingebaut. Damit soll das Dateisystem nicht angegriffen werden können und eine Verbreitung von Viren unterbunden werden. Wie die Realität zeigt, sind diese Methoden allerdings noch nicht ganz ausgereift. Es werden ständig neue Sicherheitslücken entdeckt und behoben.

Um das Arbeiten mit einem Programm in heterogenen und verteilten Netzen zu ermöglichen und Hard- und Software-unabhängig zu sein, wurde Java plattform- und betriebssystemunabhängig gestaltet. Dadurch können Anwendungen auf verschiedenen Systemen laufen, ohne verändert werden zu müssen.

Durch Unterstützung von Threads wird die Ausführgeschwindigkeit der Anwendungen gesteigert, da Teilprozesse, wie sie in multimedialen Applikationen durch Einbinden von Ton und Bildern vorkommen, parallel ausgeführt werden. Wäre beispielsweise der Link Test ein Bestandteil von WebLSD, so könnte während dessen Ausführung nicht mit dem Werkzeug gearbeitet werden, bis der Test beendet ist. Ein eigener Thread jedoch, würde hier ein weiterarbeiten ermöglichen und somit eine sinnvolle Nutzung der Wartezeit gestatten. Zusätzlich läuft ein *automatic garbage collector* als Thread im Hintergrund, der eine niedrige Priorität besitzt und benötigten Speicherplatz zur Verfügung stellt. Das kann unter Umständen auch zu einer schnelleren Ausführgeschwindigkeit beitragen.

A.2 Klassenbibliothek

Aufgrund des objektorientierten Konzepts von Java können auch hier die Vorteile dieser Programmiertechnik ausgenutzt werden, mit der die Erstellung modularer Programme und die Vererbung von Code eine Erleichterung der eigenen Arbeit darstellen. Java stellt eine große Anzahl von Klassen zur Verfügung, die je nach Funktionen in eigene Pakete, sogenannte Packages, zusammengefaßt sind.

Folgende Packages können verwendet werden:

- `java.lang` : Hauptelemente der Sprache, die automatisch eingebunden werden
- `java.util` : grundlegende Datenstrukturen (z.B.: Stack, Datum)
- `java.io` : I/O-Methoden (z.B.: Streams, Dateien)
- `java.net` : Netzwerkunterstüzung (z.B.: Sockets, ftp)
- `java.awt` : graphische Bestandteile (z.B.: Scrollbars, Menüs)
- `java.applet` : spezielle Appletfunktionen und -eigenschaften

Um auf die Eigenschaften und Funktionen einer Klasse in einem bestimmten Package zuzugreifen, muß diese Klasse erst importiert werden. Falls man z.B. Laufleisten im eigenen Programm verwenden will und dabei auf die `Scrollbar`-Klasse zurückgreift, so muß diese Klasse, die ein Bestandteil des *abstract windowing toolkit* Package `awt` ist, auf folgende Weise importiert werden:

```
import java.awt.Scrollbar
```

Um auf die Prozeduren der `Scrollbar`-Klasse zuzugreifen, muß erst eine Instanz der Klasse gebildet werden:

```
Scrollbar s = new Scrollbar();
```

Mit `s.prozedur()` können dann die Methoden der Klasse verwendet werden.

A.3 Hello World

Sowohl selbständige Applications als auch Applets für WWW-Anwendungen können mit Java leicht programmiert werden. Applets sind in Java erstellte Programme, die, in WWW-Seiten eingebunden, über das WWW heruntergeladen und von einem Browser auf dem Rechner des Benutzers ausgeführt werden. Applications sind Programme, die zu ihrer Ausführung keinen Browser benötigen, sondern mittels eines Interpreters selbständig auf jedem Rechner laufen. Sie haben im Gegensatz zu Applets auch Zugang zum Dateisystem eines Rechners, während das bei Applets unterbunden wurde. Damit wird einem Programm aus dem Netz keine Möglichkeit gegeben, Schäden auf dem jeweiligen Rechner anzurichten.

Ein Java-Programm kann entweder ein Applet oder eine Application oder auch beides sein. Das klassische Beispielprogramm *Hello World* soll das exemplarisch vorstellen. Das erste Programm ist eine selbständige Application:

```
class HelloWorld {
    public static void main (String args[]) {
        System.out.println("Hello World!");
    }
}
```

Das Programm muß unter `HelloWorld.java` abgespeichert werden, nach dem Namen der Klasse, die definiert wird. Der Rest des Programms, der Programmkörper, besteht hier aus einer `main`-Routine, die wie in C oder C++ auch, als erste beim Start aufgerufen wird. Dieses Programm gibt die Zeile `Hello World!` am Bildschirm aus.

Um es als Applet zu verwenden, muß es folgendermaßen geändert werden:

```
import java.awt.Graphics;

public class HelloWorldApplet extends java.applet.Applet {
    public void paint(Graphics g) {
        g.drawString("Hello World!", 5, 25);
    }
}
```

In diesem Fall muß es unter `HelloWorldApplet.java` abgespeichert werden. Wichtig ist, daß jedes Applet die Signatur

```
public class appletClass extends java.applet.Applet {
    ...
}
```

am Anfang als Erweiterung der Klassendefinition stehen hat. Dabei wird eine Unterklasse der `Applet`-Klasse gebildet. Diese stellt alle Eigenschaften, die für ein Applet typisch sind, zur Verfügung. Hier wird z.B. festgelegt, daß ein Applet in einem Browser abläuft oder daß es Elemente einer GUI darstellen kann, die wiederum in dem `AWT`-Package definiert sind. Deswegen wird am Anfang des Programms mit `import` die `Graphics`-Klasse, die ein Teil des `AWT`-Package ist, importiert (siehe A.2). Dadurch ist das Zeichnen und Darstellen auf dem Bildschirm erst möglich, was hier innerhalb der `paint`-Methode erfolgt.

Wie vielleicht oben auffällt, ist eine `main`-Prozedur in Applets nicht erforderlich. Hier sind, abhängig von dem Hauptanliegen des Applets, andere Prozeduren wichtig. Z.B. wird `init()` beim ersten Laden eines Applet als erstes aufgerufen, um den Anfangsstatus, die Fonts oder Parameter zu setzen. Beim Fehlen dieser Prozedur in einem Applet wird eine Default-Funktion ausgeführt, wobei dann Standardwerte eingesetzt werden.

Weitere wichtige Funktionen für Applets sind `paint()`, `start()`, `stop()` und `destroy()`. `start()` wird nach der Initialisierung aufgerufen und auch jedesmal beim neuen Zugriff auf die Web-Seite, die das Applet enthält. Ganz im Gegensatz zu `init()` kann also `start()` öfters im Leben eines Applets aufgerufen werden, während `init()` nur einmal ausgeführt wird. Wird ein Link auf eine andere Web-Seite verfolgt, so wird `stop()` aufgerufen, um das Applet zu beenden. Fehlt diese Prozedur, so wird das Applet beim Verlassen der Web-Seite weiter ausgeführt und verbraucht dabei Systemressourcen. Mit der `destroy()`-Funktion kann ein Applet nach Bedarf hinter sich aufräumen. Es kann z.B. beim eigenen Beenden oder das des Browsers, in dem es läuft, laufende Threads abstellen. In der `paint()`-Methode wird die Art und Weise festgelegt, wie das Applet auf den Bildschirm dargestellt werden soll. Diese Prozedur kann auch öfters im Leben eines Applet ausgeführt werden. Das erste Mal gleich nach der Initialisierung und dann immer wieder, wenn z.B. ein fremdes Fenster über das eigene gelegt und wieder entfernt wird.

Das Applet wird mit Hilfe eines <APPLET>-Tag in eine Web-Seite eingebunden:

```
<APPLET CODE="HelloWorldApplet.class" WIDTH=150 HEIGHT=25></APPLET>
```

Dabei wird mit CODE der Name der Klasse angegeben, die das Applet enthält und mit HEIGHT und WIDTH die Dimension des Applets.

Anhang B

Listings

B.1 WebLSD.java

```
import PositionLayout;

import documentDataStructures.*;

import java.awt.*;
import java.io.*;


// Dies ist die Hauptklasse
public class WebLSD extends Frame {

   static Setup setup;


   //Klassenvariable, die die offenen Fenster mitzaehlt
   static int anzahlOffeneFenster = 0;


   // Festlegen der min/max Fenstergroesse
   final int maxStartFensterGroesse = 750;
   final int minStartFensterGroesse = 500;
   int startFensterGroesse;


   // Menueleiste instanziieren
   static MenuBar menuBar;
   Menu m_Datei, m_Bearbeiten, m_Hilfe;
   MenuItem m_Datei_Aktualisieren, m_Datei_Schliessen,
      m_Datei_Beenden,m_Datei_NeuesFenster,
      m_Bearbeiten_SetupBearbeiten, m_Bearbeiten_SetupLaden,
      m_Bearbeiten_StrukturTesten,
      m_Hilfe_ueber, m_Hilfe_Info;
```

```
// Hauptpanel, enthaelt alles andere
Panel topPanel;

// Panel fuer die untere Leiste
Panel subPanel;


//Laufleisten
Scrollbar V_Scrollbar;
Scrollbar H_Scrollbar;


LinkStructure linkStruktur;

// die Arbeitsflaeche instanziieren
Arbeitsflaeche arbeitsflaeche;




// Konstruktor der Klasse
public WebLSD(LinkStructure linkStruktur) {

   //Aufruf des Superklassenkonstruktors
   super("WebLSD: "+linkStruktur.dokumentListe[linkStruktur.startDokument].name());

   // Klassenweiter zugriff auf die darzustellende Link Struktur
   this.linkStruktur = linkStruktur;

   // jetzt exitstiert ein Fenster mehr
   anzahlOffeneFenster++;

   // Layout des Fensters festlegen
   setLayout(new BorderLayout(10,10));



   // Menueleiste  erstellen
   menuBar = new MenuBar();


   // erstes Menu (Datei) mit Items in Menu Bar einfuegen
   m_Datei = new Menu("Datei");
   menuBar.add(m_Datei);

   m_Datei_Aktualisieren = new MenuItem("Aktualisieren");
   m_Datei.add(m_Datei_Aktualisieren);

   m_Datei_NeuesFenster = new MenuItem("Neues Fenster");
   m_Datei.add(m_Datei_NeuesFenster);

   m_Datei.add(new MenuItem("-"));
   m_Datei_Schliessen = new MenuItem("Schliessen");
```

```
m_Datei.add(m_Datei_Schliessen);

m_Datei_Beenden = new MenuItem("Beenden");
m_Datei.add(m_Datei_Beenden);


// zweites Menu (Bearbeiten) mit Items in Menu Bar einfuegen
m_Bearbeiten = new Menu("Bearbeiten");
menuBar.add(m_Bearbeiten);

m_Bearbeiten_SetupLaden = new MenuItem("Setup Laden ...");
m_Bearbeiten.add(m_Bearbeiten_SetupLaden);

m_Bearbeiten_SetupBearbeiten = new MenuItem("Setup Bearbeiten ...");
m_Bearbeiten.add(m_Bearbeiten_SetupBearbeiten);

m_Bearbeiten.add(new MenuItem("-"));
m_Bearbeiten_StrukturTesten = new MenuItem("Struktur testen ...");
m_Bearbeiten.add(m_Bearbeiten_StrukturTesten);


// Hilfe Menu erstellen
m_Hilfe = new Menu("Hilfe");
menuBar.add(m_Hilfe);

m_Hilfe_ueber = new MenuItem("Ueber ...");
m_Hilfe.add(m_Hilfe_ueber);

m_Hilfe.add(new MenuItem("-"));
m_Hilfe_Info = new MenuItem("Informationen");
m_Hilfe.add(m_Hilfe_Info);

// dieses Menu als Hilfemenu definieren
menuBar.setHelpMenu(m_Hilfe);


// Menueleiste einfuegen
this.setMenuBar(menuBar);




// einstellen der richtigen Bildschirmdimensionen

//wird das Fenster zu gross
if (linkStruktur.maxRadius > maxStartFensterGroesse) {

   // dann entsprechend verkleinern
   startFensterGroesse = maxStartFensterGroesse;
}

// oder zu klein
else if (linkStruktur.maxRadius < minStartFensterGroesse) {
```

```
      // dann entsprechend vergroessern
      startFensterGroesse = minStartFensterGroesse;
   }

   else {
      // ansonsten bei der optimalen Groesse belassen
      startFensterGroesse = linkStruktur.maxRadius;
   }

   // Laufleisten entsprechend Fenstergroesse und Arbeitsflaeche.infoSize
   //initialisieren

   // (Richtung,aktuelle Position, Ausschnittgroesse, min,max des Scrollbars)
   V_Scrollbar = new Scrollbar(Scrollbar.VERTICAL,
                               linkStruktur.maxRadius/2 -Arbeitsflaeche.infoSize,
                               startFensterGroesse,
                               1 - Arbeitsflaeche.infoSize,
                               linkStruktur.maxRadius -Arbeitsflaeche.infoSize);
   H_Scrollbar = new Scrollbar(Scrollbar.HORIZONTAL,
                               linkStruktur.maxRadius/2,
                               startFensterGroesse,
                               1,linkStruktur.maxRadius);

   // initialisieren der Arbeitsflaeche
   arbeitsflaeche = new Arbeitsflaeche(new Dimension(startFensterGroesse,
                                      startFensterGroesse+Arbeitsflaeche.infoSize),
                                    linkStruktur,
                                    new Point(linkStruktur.maxRadius/2,
                                      linkStruktur.maxRadius/2 -
                                      Arbeitsflaeche.infoSize));



   // topPanel Panel initialisieren
   topPanel = new Panel();
   topPanel.setLayout(new BorderLayout(5,5));


   // SubPanel initialisieren
   subPanel = new Panel();
   subPanel.setLayout(new BorderLayout());

   //Layout des Subpanels erstellen: Label und Laufleiste
   subPanel.add("West",new Label(" WebLSD:  WebLinkStructureDisplay   "));
   subPanel.add("Center",H_Scrollbar);


   //Layout des topPanels erstellen: Laufleisten, Arbeitsflaeche und das Subpanel
   topPanel.add("Center",arbeitsflaeche);
   topPanel.add("East",V_Scrollbar);
   topPanel.add("South",subPanel);
```

```
      // fenster am Bildschirm erzeugen
      add("Center",topPanel);
      pack();
      show();

   }


   public boolean handleEvent(Event evt) {


      //DialogFenster instanziieren
      InfoFenster infoFenster;
      FileDialog setup_LadenFenster;



      switch (evt.id) {
      case Event.SCROLL_LINE_UP:
      case Event.SCROLL_LINE_DOWN:
      case Event.SCROLL_PAGE_UP:
      case Event.SCROLL_PAGE_DOWN:
      case Event.SCROLL_ABSOLUTE:
         if (evt.target == V_Scrollbar) {
            arbeitsflaeche.ty = ((Integer)evt.arg).intValue();
            arbeitsflaeche.repaint();
         }
         if (evt.target == H_Scrollbar) {
            arbeitsflaeche.tx = ((Integer)evt.arg).intValue();
            arbeitsflaeche.repaint();
         }
         break;
      case Event.ACTION_EVENT:
         if (evt.target instanceof MenuItem) {

            // kann direkt auf == abgefragt werden, da identische Variable
            if (evt.target == m_Datei_Aktualisieren) {
               arbeitsflaeche.hide();
               new WebLSD(new LinkStructure(setup.log, setup.logFiles));
               this.hide();
               this.dispose();

            }

            if (evt.target == m_Datei_NeuesFenster) {
               new WebLSD(linkStruktur);

            }

            if (evt.target == m_Datei_Schliessen) {
               if (anzahlOffeneFenster-- > 1) this.hide();
```

```
      else System.exit(0);
   }
   if (evt.target == m_Datei_Beenden) System.exit(0);



   if (evt.target == m_Bearbeiten_SetupLaden) {
      setup_LadenFenster =
         new FileDialog (this,
                         "Setup Laden ...",
                         FileDialog.LOAD);
      setup_LadenFenster.show();
      String dateiName =  setup_LadenFenster.getDirectory()
                         + setup_LadenFenster.getFile();

      if (!dateiName.equals("nullnull")) {
         setup.load(dateiName);

         setup = new Setup(this, "SetupBearbeiten ...", true);
         setup.pack();
         setup.show();
      }
      arbeitsflaeche.repaint();
   }

   if (evt.target == m_Bearbeiten_SetupBearbeiten) {
      setup = new Setup(this, "Setup Bearbeiten ...", true);
      setup.pack();
      setup.show();
   }

   if (evt.target == m_Bearbeiten_StrukturTesten) {

      (new ExternalProgram(this, setup.progName)).start();
   }



   if (evt.target == m_Hilfe_ueber) {
      String[] anzeige = new String[6];
      anzeige[0] = "Technische Universitaet Muenchen";
      anzeige[1] = " Institut fuer Informatik ";
      anzeige[2] = " ";
      anzeige[3] = "WebLinkStructureDisplay";
      anzeige[4] = "implementiert von Diana Stricker und Albert Euba";
      anzeige[5] = "Fragen an: euba@informatik.tu-muenchen.de";
      infoFenster = new InfoFenster(this,
                     "Hilfe Ueber ...",
                     true,
                     anzeige);
   }

   if (evt.target == m_Hilfe_Info) {
```

```
                String[] anzeige = new String[1];
                anzeige[0] = "Text folgt nach Erstellung der Ausarbeitung";
                infoFenster = new InfoFenster(this, "Informationen",
                                              true, anzeige);
            }
         }
         break;


      }
      return super.handleEvent(evt);

   }



   public static void main (String args[]) {

      // Standardsetup Laden
      setup = new Setup(new Frame(),"",true);
      setup.load("WebLSD.ini");
      new WebLSD(new LinkStructure(setup.log, setup.logFiles));
   }

}





// Diese Klasse startet einen LinkTest in eigenem Prozess
class ExternalProgram extends Thread {

String befehl;
WebLSD aufrufer;
String[] anzeige = new String[1];

   public ExternalProgram(WebLSD aufrufer, String befehl) {
      super(befehl);
      this.befehl = befehl;
      this.aufrufer = aufrufer;
   }

   public ExternalProgram(String befehl) {
      super(befehl);
      this.befehl = befehl;
      this.aufrufer = null;
   }

   public void run() {
      try {
         Process prozess = Runtime.getRuntime().exec(befehl);
```

```
      //Wenn stdout erfolgen soll, dann auskommentieren
      //   InputStream iss = prozess.getInputStream();
      //   DataInputStream input =new DataInputStream(iss);
      //   String instring;
      //
      //   while ((instring = input.readLine()) != null) {
      //      System.out.println(instring);
      //   }

         try { prozess.waitFor();}
         catch(InterruptedException e) { System.out.println("InterruptedException");}

         if (prozess.exitValue() == 0)
            anzeige[0] = " Ausfuehrung von " +befehl+ " beendet";
         else
            anzeige[0] = " Fehler bei Ausfuehrung von " + befehl;
      }
      catch (IOException e) {
         System.out.println("IOException bei aufuehrung von "+ befehl);
      }


      if (aufrufer != null) {
         new InfoFenster(aufrufer,"Externer Prozess beendet", true, anzeige);
      }
   }



}
```

```
// Auf der Arbeitsflaeche werden die Icons und deren Verbindungen dargestellt
class Arbeitsflaeche extends Panel {


   // Variablen zum Scrollen
   int tx;
   int ty;

   // Hoehe des infoPanels
   final static int infoSize = 40;
```

```
// Panel und Label fuer die Statusinfo
Panel infoPanel;
Label dateiLabel;
Label pfadLabel;
Label serverLabel;
int lastTouchedDokument = -1;


// Popup Fenster bei Mausklick auf ein Icon
Panel Mousefield;
PopUp Auswahlpunkte;
String popUpItems[];


// Gesamtliste der darzustellenden Dokumente
LinkStructure struktur;

//Hoehe und Breite des Kaestchens zur Darstellung der Dokumente
int dokumentWidth = 70;
int dokumentHeight = 14;



// Konstruktor
Arbeitsflaeche(Dimension size, LinkStructure struktur, Point mittelpunkt) {

   // Aufruf des Superklassenkonstruktors
   super();


   // Zugriff auf Liste aller Dokumente
   this.struktur = struktur;


   // Layout der Arbeitsflaeche als PositionLayout festlegen
   setLayout(new PositionLayout(size.width,size.height));

   // Hintergrundfarbe festlegen
   setBackground(Color.white);

   // Font festlegen
   setFont(new Font("Courier", Font.PLAIN, 12));


   // Arbeitsflaeche auf Mitte scrollen
   tx = mittelpunkt.x;
   ty = mittelpunkt.y;


   // Popup Panel definieren
   Mousefield = new Panel();
   Mousefield.setLayout(new FlowLayout());
   String[] popUpItems = new String[1];
```

```
    popUpItems[0] = "Name der Seite";
    Auswahlpunkte = new PopUp();
    Mousefield.add(Auswahlpunkte);
    add("0 0",Mousefield);
    Mousefield.hide();


    // Info Label
    //initialisieren
    infoPanel = new Panel();
    infoPanel.setLayout(new GridLayout(2,3,0,0));
    infoPanel.setBackground(Color.lightGray);

    //belegen
    infoPanel.add(new Label("WWW-Server:           "));
    infoPanel.add(new Label("  Pfad:"));
    infoPanel.add(new Label("  Dateiname:"));

    dateiLabel = new Label("");
    pfadLabel  = new Label("");
    serverLabel = new Label("");
    infoPanel.add(serverLabel);
    infoPanel.add(pfadLabel);
    infoPanel.add(dateiLabel);

    //einfuegen
    add("0 0",infoPanel);

  }



  public boolean handleEvent(Event evt) {

    Point dokumentKoordinaten;

    // Laufvariable
    int dokument;

    // Ereignis auf der Arbeitsflaeche?
    if (evt.target == this) {

      // Was macht der Benutzer
      switch (evt.id) {

      // einfache Bewegung der Maus
      case Event.MOUSE_MOVE:
        // Im InfoPanel anzeigen, ueber welchem Dokument sich die Maus befindet

        // durchsuche dokumentListe, ob Mauskoordinaten passen
        for(dokument = struktur.dokumentListe.length -1;
            dokument >= 0 ;
            dokument--)   {
```

```
            dokumentKoordinaten=struktur.dokumentListe[dokument].liesKoordinaten();
            if (   // x - Bereich abpruefen
               //                  Umfeld: halbes Kaestchen und Scrollbarstaende
                ((dokumentKoordinaten.x < evt.x + dokumentWidth/2 + tx)  &&
                (dokumentKoordinaten.x  > evt.x - dokumentWidth/2 + tx)) &&
               // y - Bereich abpruefen
                ((dokumentKoordinaten.y < evt.y + dokumentHeight/2 + ty) &&
                (dokumentKoordinaten.y  > evt.y - dokumentHeight/2 + ty))) {

               // entsprechendes Dokument gefunden
               if (dokument != lastTouchedDokument) {

                  // Nur wenn Maus ueber neuem Dokument, Labels updaten
                  dateiLabel.
                     setText("  "+struktur.dokumentListe[dokument].dateiName);
                  pfadLabel.
                     setText("  "+struktur.dokumentListe[dokument].pfadName);
                  serverLabel.
                     setText(struktur.dokumentListe[dokument].serverName);

                  // und am Bildschirm darstellen
                  infoPanel.layout();

                  // dann aktuelles Dokument merken
                  lastTouchedDokument = dokument;
               }
               // Dokument gefunden, Laufschleife verlassen
               break;
            }
         }
         // Wenn aber die Maus gerade ein Dokument verlassen hat
         if ((dokument <= 0) && (lastTouchedDokument != -1)) {

            // dann Labels loeschen
            dateiLabel.setText("");
            pfadLabel.setText("");
            serverLabel.setText("");

            // und am Bildschirm darstellen
            infoPanel.layout();

            // kennzeichnen, dass kein Dokument ausgewaehlt
            lastTouchedDokument = -1;
         }
         // Handlung erledigt, switch verlassen
         break;


      // Mausklick
      case Event.MOUSE_DOWN:
         // PopUp ueber aktuellem Dokument oeffnen
```

```
// durchsuche dokumentListe, ob Mauskoordinaten passen
for(dokument = struktur.dokumentListe.length -1;
    dokument >= 0 ;
    dokument--)  {

   dokumentKoordinaten=struktur.dokumentListe[dokument].liesKoordinaten();
   if (   // x - Bereich abpruefen
       //                  Umfeld: halbes Kaestchen und Scrollbarstaende
       ((dokumentKoordinaten.x < evt.x + dokumentWidth/2 + tx)  &&
       (dokumentKoordinaten.x  > evt.x - dokumentWidth/2 + tx)) &&
       // y - Bereich abpruefen
       ((dokumentKoordinaten.y < evt.y + dokumentHeight/2 + ty) &&
       (dokumentKoordinaten.y  > evt.y - dokumentHeight/2 + ty))) {

      // entsprechendes Dokument gefunden, dann


      // erzeugen des passenden PopUps

      // loeschen des alten PopUps
      Mousefield.remove(Auswahlpunkte);

      // initialisieren der PopUps Items
      int laenge = 0;
      popUpItems = new
         String[12 + struktur.dokumentListe[dokument].vater.length];

      // Name an oberster Stelle
      popUpItems[laenge++] = struktur.dokumentListe[dokument].name();
      popUpItems[laenge++] = "";

      // Vermerk, falls Dokument fehlerhaft
      if (struktur.dokumentListe[dokument].fehlernummer != 0) {
        popUpItems[laenge++] =
           "Fehler: ";
        popUpItems[laenge++] =
           struktur.dokumentListe[dokument].fehlerText();
        popUpItems[laenge++] = "";
      }


      // Zoom anbieten, wenn Soehne vorhanden
      if (struktur.dokumentListe[dokument].sohn.length > 0) {
         popUpItems[laenge++] = "Zoom auf Unterstruktur";
         popUpItems[laenge++] = "";
      }

      // Anzeige der Vorfahren
      if (struktur.dokumentListe[dokument].vater.length != 0) {
         popUpItems[laenge++] = "Verweise auf das Dokument in:";
         for (int i = 0;
              i < struktur.dokumentListe[dokument].vater.length;
              i++) {
```

```
                    if (struktur.dokumentListe[dokument].vater[i] != -1)
                       popUpItems[laenge++] = struktur.dokumentListe[
                         struktur.dokumentListe[dokument].vater[i]].name();
                 }
                 popUpItems[laenge++] = "";
              }

              // Darstellung anbieten, wenn fehlerfrei
              if (struktur.dokumentListe[dokument].fehlernummer == 0) {
                 popUpItems[laenge++] = "Dokument anzeigen";
                 popUpItems[laenge++] = "";
              }

              // Ende Option an letzter Stelle
              popUpItems[laenge++] = "Popup schliessen";
              Auswahlpunkte=new PopUp(Mousefield, struktur, popUpItems, dokument);


              // einfuegen des PopUps
              Mousefield.add(Auswahlpunkte);
              add((evt.x-30) + " " + (evt.y-10), Mousefield);


              // Darstellung des aktuellen Layouts
              // labels loeschen
              dateiLabel.setText("");
              pfadLabel.setText("");
              serverLabel.setText("");
              layout();
              Mousefield.show();

              break;
           }

           // Wenn kein Dokument ausgewaehlt, kein PopUp
           else Mousefield.hide();
        }

        // Handlung erledigt, switch verlassen
        break;

     }
  }

  // aufruf der Superklassen-handleEvent Methode
  return super.handleEvent(evt);
}



//Zeichnen der Verbindungslinien zu allen Soehnen
void Edges(Graphics g, Dokument dokument) {
```

```
      //Punkte fuer Anfang und Ende der Verbindungslinien
      Point point1, point2;

      //Anfangskoordinaten der Linie (Vater) ermitteln
      point1 = new Point(0,0);
      point1 = dokument.liesKoordinaten();

      // Linien zu allen Soehnen ziehen
      point2 = new Point(0,0);
      for (int i = 0; i < dokument.sohn.length; i++) {

         // neue Endkoordinaten ermitteln
         point2 = struktur.dokumentListe[dokument.sohn[i]].liesKoordinaten();

         if (struktur.dokumentListe[dokument.sohn[i]].fehlernummer != 0)  {
            // i-ter Sohn fehlerhaft, Linie rot
            g.setColor(Color.red);
         }
         else {
            // Sonst linie schwarz
            g.setColor(Color.black);
         }
         g.drawLine(point1.x, point1.y, point2.x, point2.y);
      }
   }


   //Zeichnen aller Verbindungslinien durch Edges()
   public void paintEdges(Graphics g) {

      //Darstellen der Verbindungen aller Dokumente
      for (int i = 0; i < struktur.dokumentListe.length; i++) {
            Edges(g, struktur.dokumentListe[i]);
      }
   }

   public void paintNodes(Graphics g) {

      Point koordinatenRect;
      Point koordinatenString;
      int xRect, yRect, xString, yString;

      // Variablen zur Darstellung der Dokumentbezeichnung
      String darzustellenderText;
      int maxDarstellbarerString = 9;


      for (int i = 0; i < struktur.dokumentListe.length; i++) {
         //Koordinaten des Kaestchens
         koordinatenRect = struktur.dokumentListe[i].liesKoordinaten();
         xRect = koordinatenRect.x - (dokumentWidth/2);
         yRect = koordinatenRect.y - (dokumentHeight/2);
```

```
      //Darstellen der Kaestchen

      //Farbe der Kaestchen festlegen
      if (struktur.dokumentListe[i].fehlernummer != 0) {
         // Fehlerhaftes Dokument, dann rot
         g.setColor(Color.red);
      }
      else if (! struktur.dokumentListe[i].serverName.equals(
                 struktur.dokumentListe[struktur.startDokument].serverName)) {
         // Fremder Server, dann gelb
         g.setColor(Color.yellow);
      }
      //sonst gruen
      else g.setColor(Color.green);

      // Kaestchen ausfuellen
      g.fillRect(xRect, yRect, dokumentWidth, dokumentHeight);

      // schwarzen Rahmen zeichnen
      g.setColor(Color.black);
      g.drawRect(xRect, yRect, dokumentWidth, dokumentHeight);


      //Darzustellenden Namen des Dokuments ermitteln

      // Namen erfragen
      darzustellenderText = struktur.dokumentListe[i].name();

      // passt der ganze Name in nicht das Feld
      if (!(darzustellenderText.length() <= maxDarstellbarerString)) {
         // dann Namen entsprechend verkuerzen
         darzustellenderText =
            darzustellenderText.substring(0,maxDarstellbarerString);
      }

      //Koordinaten, an denen der Text geschrieben wird
      koordinatenString = struktur.dokumentListe[i].liesKoordinaten();

      // Mittelpunkt verschieben, damit Text in die Box passt
      xString = koordinatenString.x - 31;
      yString = koordinatenString.y + 5;
      // Text schreiben
      g.drawString(darzustellenderText, xString, yString);
   }
}


// hier erfolgt die Darstellung
public void paint(Graphics g) {

   // ausrichtung nach Scrollbars
   g.translate(-tx, -ty);
```

```
      // Darstellung der Struktur
      paintEdges(g);
      paintNodes(g);
   }
}





// Popup fuer die Arbeitsflaeche
class PopUp extends Choice {

   // Instanz der obigen Arbeitsflaeche
   LinkStructure struktur;
   Panel panel;

   // index des aktuellen Dokuments
   int dokument;




   // Konstruktor fuer leeres PopUp
   PopUp() {
      super();

      // ein Item muss vorhanden sein, sonst Fehler
      addItem(" ");

   }



   // eigentlich verwendeter Konstruktor der Klasse
   PopUp(Panel panel, LinkStructure struktur, String[] item, int dokument) {

      // Aufruf des Superklassenkonstruktors
      super();

      // Verwendung der jeweiligen Variablen
      this.struktur = struktur;
      this.panel = panel;
      this.dokument = dokument;

      // initialisiere Itemliste
      for(int i = 0; i < item.length; i++) {

         if (item[i] != null) addItem(item[i]);
      }
   }
```

```
   public boolean handleEvent(Event evt) {

      if (evt.target == this) {

         if (getSelectedItem().equals("Zoom auf Unterstruktur")) {

            //System.out.println("Zoom ...");
            new WebLSD( new LinkStructure(struktur, dokument));
         }

         if (getSelectedItem().equals("Dokument anzeigen")) {

            if (Setup.browser != "") {
               (new ExternalProgram(Setup.browser + " " +
                                    struktur.dokumentListe[dokument].url())).start();
            }
         }

         // nach Auswahl PopUp loeschen
         panel.hide();
      }


   // aufruf der Superklassen-handleEvent Methode
   return super.handleEvent(evt);

   }

}





// Setup Konfigurationen
class Setup  extends Dialog {

   //Variablen fuer das Layout
   Button ok;
   Button save;
   Button cancel;

   Panel topPanel;
   Panel subPanel;


   // Auswahl der auszuwertenden LOG Files
   Choice logMenu;

   final int all = 0;
   final int tested = 1;
   final int own = 2;
```

```
static int logFiles = -1;


// gemeinsamer Name der Files
TextField logField;
static String log;

// Name des Programms
TextField progNameField;
static String progName;


// absoluter Name des zu verwendenden Webbrowsers
TextField browserField;
static String browser;


// Name des ini Files
TextField iniNameField;
static String iniName;


// Setup laden
public void load(String setupName) {

   // Name der Setup Datei abspeichern
   iniName = setupName;

   try {
      try {
         DataInputStream setupDatei = new
            DataInputStream(new FileInputStream(setupName));

         // fuer alle Datensaetze: einlesen
         logFiles = Integer.parseInt(setupDatei.readLine());
         log      = setupDatei.readLine();
         progName = setupDatei.readLine();
         browser  = setupDatei.readLine();

         // .ini schliesen, wenn erledigt
         setupDatei.close();
      }
      catch (FileNotFoundException e) {
         // .ini Datei nicht gefunden, Standard Werte setzen
         System.out.println("Datei "+ setupName +" nicht gefunden");
         logFiles = all;
         log      = "LinkTest";
         progName = "LinkTest";
         browser  = "";
      }
   }
   catch (IOException e) {
```

```
        System.out.println("Fehler beim lesen der Datei "
                          + setupName
                          +" aufgetreten");
        // Exception vor oder nach dem Einlesen der Werte
        if (logFiles == -1)   logFiles = all;
        if (log == null)      log      = "LinkTest";
        if (progName == null) progName = "LinkTest";
        if (browser == null)  browser  = "";
     }

  }

  //Konstruktor fuer Setup bearbeiten
  Setup(Frame fr, String strg, boolean bool) {

     //Aufruf des Superklassenkonstruktors
     super(fr, strg, bool);

     //Dialogbox fuer Setup Bearbeiten... gestalten
     setLayout(new BorderLayout());

     //Hauptfenster fuer Information gestalten
     topPanel = new Panel();
     topPanel.setLayout(new GridLayout(6,2));

     topPanel.add(new Label("Name des Link Test Programms: "));
     topPanel.add(progNameField = new TextField(progName,25));

     topPanel.add(new Label("Name der LOG - Dateien:"));
     topPanel.add(logField = new TextField(log, 25));

     topPanel.add(new Label("Dokumentauswahl:"));
     logMenu = new Choice();
     logMenu.addItem("alle Dokumente");
     logMenu.addItem("fehlerfreie Dokumente");
     logMenu.addItem("getestete Dokumente");
     logMenu.select(logFiles);
     topPanel.add(logMenu);

     topPanel.add(new Label("WWW-Browser: "));
     topPanel.add(browserField = new TextField(browser,25));

     topPanel.add(new Label(""));
     topPanel.add(new Label(""));

     topPanel.add(new Label("Setup speichern als: "));
     topPanel.add(iniNameField = new TextField(iniName,25));



     //Unterfenster fuer Button erzeugen
     subPanel = new Panel();
     subPanel.setLayout(new FlowLayout(FlowLayout.CENTER,50,50));
```

```
      subPanel.add(ok = new Button ("OK"));
      subPanel.add(cancel = new Button("Cancel"));
      subPanel.add(new Label(" "));
      subPanel.add(save = new Button("Save"));

      //das gesamte Layout in DialogBox einfuegen
      add("Center",topPanel);
      add("South",subPanel);

      //Anzeigen der Dialog-Box muss ausserhalb erfolgen

   }


   public void save() {

      try {

         DataOutputStream setupDatei = new
            DataOutputStream(new FileOutputStream(iniName));
         //System.out.println("Schreiben des Files: "+ iniName);

         // fuer alle Datensaetze: abspeichern
         setupDatei.writeBytes(String.valueOf(logFiles)+"\n");
         setupDatei.writeBytes(log+"\n");
         setupDatei.writeBytes(progName+"\n");
         setupDatei.writeBytes(browser+ "\n");

         // .ini schliesen, wenn erledigt
         setupDatei.close();
      }
      catch (IOException e) {
         System.out.println(iniName +" konnte nicht korrekt gespeichert werden");
      }

   }


   public boolean handleEvent (Event evt) {
      switch (evt.id) {
      case Event.ACTION_EVENT:

         //cancel, dann verwerfen
         if (evt.target == cancel) {
            this.hide();
         }

         // ansonsten uebernehmen
         else {
            progName = progNameField.getText();
            log      = logField.getText();
            logFiles = logMenu.getSelectedIndex();
            browser  = browserField.getText();
```

```
            iniName  = iniNameField.getText();

            this.hide();

            // bei save noch abspeichern
            if (evt.target == save) this.save();

         }


         break;
      }
      return super.handleEvent(evt);
   }

}





// Fenster fuer kurze Informationen
class InfoFenster extends Dialog {

   Button ok;
   Panel topPanel;
   Panel subPanel;

   //Konstruktor fuer InfoFenster
   InfoFenster(Frame fr, String name, boolean bool, String[] inhalte) {

      //Aufruf des Superklassenkonstruktors
      super(fr, name, bool);

      //Dialogbox fuer Hilfe Ueber... gestalten
      setLayout(new BorderLayout());

      //Hauptfenster fuer Information gestalten
      topPanel = new Panel();
      topPanel.setLayout(new GridLayout(inhalte.length + 2, 1));
      topPanel.add(new Label(""));
      for (int i = 0; i< inhalte.length; i++) {
         topPanel.add(new Label(inhalte[i],Label.CENTER));
      }

      //Unterfenster fuer Button erzeugen
      subPanel = new Panel();
      subPanel.setLayout(new FlowLayout());
      subPanel.add(ok = new Button ("OK"));

      //das gesamte Layout in DialogBox einfuegen
      add("Center",topPanel);
```

```
      add("South",subPanel);

      //Anzeigen der DialogBox
      this.pack();
      this.show();

   }


   public boolean handleEvent (Event evt) {
      switch (evt.id) {
      case Event.ACTION_EVENT:
         if (evt.target instanceof Button) {
            if (evt.target == ok) this.hide();
         }
         break;
      }
      return super.handleEvent(evt);
   }

}
```

B.2 documentDataStructures Package

```
package documentDataStructures;


public class BasisDokument {

        // URL des Dokuments aufgeschluesselt
        public String serverName = "";
        public String pfadName = "";
        public String dateiName = "";

        // Zugriffsmethode auf das Dokument, z.B. FTP...
        public String zugriffsMethode;


        // Nummer, falls Fehler mit Dokument aufgetreten
        public int fehlernummer = 0;

        // Name des Betreuers, falls vorhanden
        public String betreuer;


        // ermittelt den Namen des Dokuments
        public String name() {
```

```
        // existiert ein Dateiname
        if    (! dateiName.equals(""))  return dateiName;

        // oder nur ein Pfadname
        else if  (! pfadName.equals(""))   return pfadName;

        // oder vielleicht nur eine Serverbezeichnung
        else  return serverName;
}

//ermittelt die URL des Dokuments
public String url() {
        return zugriffsMethode + "://" + serverName + pfadName + dateiName;
}


//liefert ausfuehrliche Fehlerbeschreibung
public String fehlerText() {
        String fehler = "";
        switch (fehlernummer) {

                case 400: fehler =  "Falsche URL Syntax";
                                break;
                case 401: fehler =  "keine Zugriffsberechtigung";
                                break;
                case 402: fehler =  "Payment Required";
                                break;
                case 403: fehler =  "Zugriff verweigert";
                                break;
                case 404: fehler =  "Dokument nicht gefunden";
                                break;
                case 405: fehler =  "falsche Zugriffsmethode";
                                break;
                case 406: fehler =  "Accept Klausel falsch";
                                break;
                case 407: fehler =  "Password benoetigt";
                                break;
                case 408: fehler =  "Timeout";
                                break;
                case 409: fehler =  "beim abspeichern";
                                break;
                case 410: fehler =  "Dokument wurde geloescht";
                                break;
                case 411: fehler =  "Content Length falsch";
                                break;
                case 412: fehler =  "Unless erfuellt";
                                break;

                case 500: fehler =  "Serverfehler";
                                break;
                case 501: fehler =  "Zugriffsmethode nicht unterstuetzt";
                                break;
                case 502: fehler =  "Proxyfehler";
```

```
                                        break;
                        case 503: fehler =  "Server nicht erreichbar";
                                        break;
                        case 504: fehler =  "Proxy Timeout";
                                        break;

                        // LinkTest interne Fehler
                        case 600: fehler =  "Socket nicht erstellbar";
                                        break;
                        case 601: fehler =  "Rechnerverbindung unmoeglich";
                                        break;
                        case 602: fehler =  "Domain nicht erreichbar";
                                        break;
                        case 603: fehler =  "Leere Seite";
                                        break;

                }
                return fehler;
        }

}
```

```
package documentDataStructures;

import documentDataStructures.BasisDokument;

import java.util.Vector;

public class VollDokument extends BasisDokument {

        public String name;

        public String[] vaterName;

        public Vector sohn  = new Vector(1,1);
        public Vector vater = new Vector(1,1);

}
```

```
package documentDataStructures;

import documentDataStructures.BasisDokument;

import java.awt.Point;
```

```
public class Dokument extends BasisDokument {

        // Liste der Indizes von direkten Vorfahren und Nachfolgern
        public int[] vater;
        public int[] sohn;


        // Ort, an dem das Dokument dargestellt wird
        Point Darstellungsort = new Point(0,0);


        // Kontrollvariable zur Erkennung erledigter Dokumente
        boolean visited = false;



        // setzen der Kontrollvariable
        public void setVisited() {
                this.visited = true;
        }

        // loeschen der Kontrollvariable
        public void clearVisited() {
                this.visited = false;
        }

        // abfragen der Kontrollvariable
        public boolean isVisited() {
                return this.visited;
        }




        // setzen der Koordinaten
        public void setzeKoordinaten(Point Koordinaten) {
                this.Darstellungsort.x = Koordinaten.x;
                this.Darstellungsort.y = Koordinaten.y;
        }

        // abfragen der Koordinaten
        public Point liesKoordinaten() {
                Point Koordinaten = new
                   Point(this.Darstellungsort.x,this.Darstellungsort.y);
                return Koordinaten;
        }


        // kopieren eines Dokuments
        public Dokument copy() {

                Dokument Kopie = new Dokument();
```

```
                Kopie.zugriffsMethode = new String(this.zugriffsMethode);

                Kopie.dateiName  = new String(this.dateiName);
                Kopie.pfadName   = new String(this.pfadName);
                Kopie.serverName = new String(this.serverName);

                Kopie.sohn = new int[this.sohn.length];
                for (int i = 0; i < this.sohn.length; i++) {
                        Kopie.sohn[i] = this.sohn[i];
                }

                Kopie.vater = new int[this.vater.length];
                for (int i = 0; i < this.vater.length; i++) {
                        Kopie.vater[i] = this.vater[i];
                }

                if (this.betreuer != null) Kopie.betreuer = new String(this.betreuer);

                Kopie.fehlernummer = this.fehlernummer;

                return Kopie;
        }


}
```

```
package documentDataStructures;

import java.awt.Point;
import java.util.Vector;
import java.io.*;

import documentDataStructures.*;


public class LinkStructure {

   //Variablen fuer den Zugriff auf die Daten

   // Dies ist die Liste aller Dokumente
   public Dokument[] dokumentListe;

   // Index der Basis URL
   public int  startDokument;



   // Variablen fuer die Darstellung
```

```
// benoetigte Flaeche
public int maxRadius = 0;

// erweiterung des Radius fuer jeden Kreis
final int addRadius = 90;

// maximaler Winkel fuer die Soehne
final double MaxWinkel = (9*Math.PI)/18;



// Konstanten fuer die Bearbeitung der Log Dateien

// Kennzeichen eines Datensatzes im File
final String Datensatzkennzeichen = "Intern --:";

// Feldseparator im Datensatz
final String feldSeparator = ";";

// Kennzeichen fuer das Ende des Vaternamens
final String endeDesVaternamens = ":";

// Kennzeichen fuer einen weiterne Vaternamen
final String naechsterVater = "|";

// Namenserweiterungen fuer die LOG Dateine
final String erledigtExtension   = ".ERL.LOG";
final String ausserhalbExtension = ".NEU.LOG";
final String fehlerExtension     = ".ERR.LOG";


// Zur Berechnung benoetigte Klassenglobale Variablen

// zum zwischenspeichern der eingelesenen Dokumente
Vector gefundeneDokumente = new Vector(50,50);

//Zaehler zum Datei einlesen und ermitteln einer neuen Struktur
int letzterEintrag = 0;




// Konstruktor bei Aufruf durch Zoom
public LinkStructure(LinkStructure ausgangsstruktur, int startIndex) {

   // rekursives ermitteln den neuen Struktur
   dokumentListe = getLinkStructure(new int[ausgangsstruktur.dokumentListe.length],
                                    ausgangsstruktur,
                                    startIndex);

   // berechnen der Koordinaten in der Liste zur Darstellung
   Display();
```

```
}


// Konstruktor bei erstmaligem Aufruf
public LinkStructure(String Dateiname, int selectLogFiles) {


   // alle angeforderten Logfiles einlesen
   dateiEinlesen(Dateiname+erledigtExtension);

   switch (selectLogFiles) {

   case 0:
      dateiEinlesen(Dateiname+ausserhalbExtension);
      dateiEinlesen(Dateiname+fehlerExtension);
      break;
   case 1:
      dateiEinlesen(Dateiname+ausserhalbExtension);
      break;
   case 2:
      dateiEinlesen(Dateiname+fehlerExtension);
      break;
   }

   // erzeugung des Link - "Graphen" und speichern in dokumentListe
   dokumentListe = getLinkStructure(gefundeneDokumente);

   // berechnen der Koordinaten in der Liste zur Darstellung
   Display();

}




void dateiEinlesen(String Dateiname) {

   String gelesen = "";
   String Datensatz;


   // Einlesen der Datei
   try {
      try {
         // oeffnen des Eingabestromes aus der Datei mit den erledigten Dokumenten
         DataInputStream Datei = new
               DataInputStream(new FileInputStream(Dateiname));


         //System.out.println("Bearbeitung des Files: "+Dateiname);

         // solange noch Zeichen in der Datei vorhanden
         while (gelesen != null)  {
```

```
          //einlesen einer Zeile der aktuellen Datei
          gelesen = Datei.readLine();
          if (gelesen == null) break;

          // test ob die Zeile einen Datensatz enthaelt
          if (gelesen.startsWith(Datensatzkennzeichen)) {

             // wenn ja entsprechenden Datensatz uebernehmen
             Datensatz = gelesen.substring(Datensatzkennzeichen.length() + 1);
             gefundeneDokumente.addElement(getEntry(Datensatz));

             //ist Dokument eine Mail adresse
             if (((VollDokument)gefundeneDokumente.elementAt(
                      letzterEintrag)).zugriffsMethode.equals("MAILTO")) {
                // dann verwerfen
                gefundeneDokumente.removeElementAt(letzterEintrag);
             }
             // ansonsten Anzahl erhoehen
             else letzterEintrag++;
          }

       }

       // nach Beendigung Datei schliessen
       Datei.close();
    }
    catch (FileNotFoundException e) {
       // Fehlerbehandlung falls Datei nicht existiert
       System.out.println("Achtung:  Datei "+Dateiname+" nicht gefunden");
    }
 }

 catch (IOException e) {
    // Fehlerbehandlung falls Datei nicht gelesen werden konnte
    System.out.println("Fehler: Probleme beim auslesen der Datei aufgetreten");
 }

}



// diese Methode loescht alle visited Eintraege
public void clearVisited() {
   try {
      for(int i = 0; i < dokumentListe.length; i++) {
         dokumentListe[i].clearVisited();
      }
   }
   // bei falscher benutzung tritt ein Fehler auf
   catch (NullPointerException e) {
      System.out.println("clearVisited() erst nach Initialisierung
                          von dokumentListe benutzen");
```

```
   }
}





// diese Methode wandelt einen Datensatzstring in ein Volldokument um
VollDokument getEntry(String input) {
   // Variable zur aufnahme der Information
   VollDokument aktuellesDokument = new VollDokument();
   // Hilfsvariable zum zerlegen des inputstrings
   String[] eintragsliste = new String[4];
   // Hilfsvariable zum zerlegen der Vaeterinformation
   Vector Vaeter = new Vector(1,1);
   // Indizes zum zerlegen des inputstrings
   int lastIndex = 0;
   int nextIndex;

   // zerlegen des Inputstrings in einzelne Eintraege
   for(int i = 0; i < 4; i++) {
      // ermittle Index des naechsten Separators
      nextIndex = input.indexOf(feldSeparator,lastIndex);
      // Index gueltig ?
      if (nextIndex == -1) {
         // wenn nein, Schleife mit Fehlermeldung beenden
         System.out.println("Datensatz im File fehlerhaft");
         i = 5;
      }
      // sonst Feldeintrag uebernehmen
      else {
         try {
            eintragsliste[i] = input.substring(lastIndex,nextIndex);
            // und naechsten Startindex weiterschalten
            lastIndex = nextIndex + 1;
         }
         // dieser Fehler sollte nie auftreten,
         // muss aber syntaktisch abgefangen werden
         catch(StringIndexOutOfBoundsException e) {
            System.out.println("Datensatz im File fehlerhaft");
            i = 5;
         }
      }
   }
   // aktuelles Dokument mit ermittelten Werten belegen
   aktuellesDokument.name = eintragsliste[0];
   aktuellesDokument.zugriffsMethode = eintragsliste[1];
   // eintrag fuer Fehler oder Betreuer
   try {
      if (! eintragsliste[3].equals("")) {
         // Fehlernummer abspeichern
         aktuellesDokument.fehlernummer = Integer.parseInt(eintragsliste[3]);
      }
```

```
    }
    catch (NumberFormatException e) {
       //wenn es keine Fehlernummer war, enthaelt der Eintrag den Betreuer
       aktuellesDokument.betreuer = eintragsliste[3];
       // und das Dokument ist fehlerfrei
       aktuellesDokument.fehlernummer = 0;
    }

    // ermittlung der Vaeter
    try {
       // solange noch Vaeter vorhanden
       while (eintragsliste[2] != null) {

          // gibt es einen Vater ?
          nextIndex = eintragsliste[2].indexOf(endeDesVaternamens);
          if (nextIndex == -1) {
             // wenn nein, Schleife mit Fehlermeldung beenden
             System.out.println("Datensatz zu "
                                +aktuellesDokument.name
                                +" im File fehlerhaft");
             eintragsliste[2] = null;
          }
          // sonst den Namen des Vaters bestimmen
          else   Vaeter.addElement(eintragsliste[2].substring(0,nextIndex));

          // folgt ein weiterer Vater ?
          nextIndex = eintragsliste[2].indexOf(naechsterVater);
          if (nextIndex != -1) {
             // dann eintragsliste[2] um bisherigen Vater verkuerzen
             eintragsliste[2] = eintragsliste[2].substring(nextIndex + 1);
          }
          // sonst Schleife beenden
          else   eintragsliste[2] = null;
       }
    }
    catch (StringIndexOutOfBoundsException e) {
       // dieser Fehler sollte auch nie vorkommen
       System.out.println("Datensatz zu "+aktuellesDokument.name
                          +" im File fehlerhaft");
    }

    // gefundene Vaeter in aktuelles Dokument eintragen
    aktuellesDokument.vaterName = new String[Vaeter.size()];
    Vaeter.copyInto(aktuellesDokument.vaterName);

    // ermitteltes Ergebnis zurueckgeben
    return aktuellesDokument;
 }


 //erzeugt aus einer vorgegebenen Liste eine neue beginnend mit startIndex
 Dokument[] getLinkStructure(int[] neueNamen,
                             LinkStructure alteStruktur,
```

```
                              int startIndex) {

      Vector neueStruktur = new Vector(50,50);

      Dokument[] erzeugteListe;


      // markieren aller Indizes als noch nicht in neuerStruktur enthalten
      for (int i = 0; i < neueNamen.length; i++) {
         neueNamen[i] = -1;
      }

      // markieren aller Dokumente als Unbesucht
      alteStruktur.clearVisited();

      // erzeuge rekursiv die Unterstruktur ausgehend von startIndex
      createSubtree(neueNamen, neueStruktur, alteStruktur, startIndex);

      // uebernahme der Struktur in Rueckgabevariable
      erzeugteListe = new Dokument[neueStruktur.size()];
      neueStruktur.copyInto(erzeugteListe);


      // berichtigen der Vater/Sohn Indizes
      for (int i = 0; i < erzeugteListe.length; i ++) {

         for (int j = 0; j < erzeugteListe[i].sohn.length; j++) {

            // jeden Sohnindex mit neuer Nummer versehen
            erzeugteListe[i].sohn[j] = neueNamen[erzeugteListe[i].sohn[j]];
         }

         for (int j = 0; j < erzeugteListe[i].vater.length; j++) {

            // jeden Vaterindex mit neuer Nummer versehen achte auf undef. Vaeter
            if (erzeugteListe[i].vater[j] != -1) {
               erzeugteListe[i].vater[j] = neueNamen[erzeugteListe[i].vater[j]];
            }
         }
      }



      // Uebertragung begann mit erstem Dokument
      startDokument = 0;

      // bei diesem sind die Vaeter ueberfluessig
      erzeugteListe[startDokument].vater = new int[0];

      return erzeugteListe;

   }
```

```
// rekursives erzeugen einer Zoomstruktur
void createSubtree(int[] neueNamen,
                   Vector neueStruktur,
                   LinkStructure alteStruktur, int index) {

   // Dokument als besucht markieren
   alteStruktur.dokumentListe[index].setVisited();

   // Dokument in die neue Struktur kopieren
   neueStruktur.addElement(alteStruktur.dokumentListe[index].copy());

   // index des Dokuments in der neuen Struktrur speichern
   neueNamen[index] = letzterEintrag++;

   // alle unbesuchten Soehne rekursiv abklappern
   for (int i = 0; i < alteStruktur.dokumentListe[index].sohn.length; i ++) {
      if (! alteStruktur.dokumentListe[
               alteStruktur.dokumentListe[index].sohn[i]].isVisited()) {
         createSubtree(neueNamen, neueStruktur,
                       alteStruktur, alteStruktur.dokumentListe[index].sohn[i]);
      }
   }

}

//erzeugt aus den gefundenen Dokumenten die gesamtListe
Dokument[] getLinkStructure(Vector gefundeneDokumente) {
   int[] hilfsliste;

   // instanziieren der Liste fuer die bearbeiteten Dokumente
   Dokument[] bearbeiteteDokumente = new Dokument[gefundeneDokumente.size()];

   // uebernahme aller Variablen
   for(int i = 0; i < gefundeneDokumente.size(); i++) {
      // neuer Eintrag
      bearbeiteteDokumente[i] = new Dokument();

      // Aufspaltung der URL
      String name = ((VollDokument)gefundeneDokumente.elementAt(i)).name;
      try {
         bearbeiteteDokumente[i].serverName =
            name.substring(0,name.indexOf("/")+1);
         bearbeiteteDokumente[i].pfadName   =
            name.substring(name.indexOf("/")+1,name.lastIndexOf("/")+1);
         bearbeiteteDokumente[i].dateiName  =
            name.substring(name.lastIndexOf("/")+1);
      }
      catch (StringIndexOutOfBoundsException e) {
         // ist kein Fehler, sondern Dateiname fehlt oder URL ist Server
      }

      bearbeiteteDokumente[i].zugriffsMethode =
```

```
         ((VollDokument)gefundeneDokumente.elementAt(i)).zugriffsMethode;
      bearbeiteteDokumente[i].betreuer =
         ((VollDokument)gefundeneDokumente.elementAt(i)).betreuer;
      bearbeiteteDokumente[i].fehlernummer =
         ((VollDokument)gefundeneDokumente.elementAt(i)).fehlernummer;

      bearbeiteteDokumente[i].sohn  = new int[0];
      bearbeiteteDokumente[i].vater = new int[0];


      // wenn TOP als Vatereintrag steht, ist dies das StartDokument
      if (((VollDokument)gefundeneDokumente.elementAt(i)).vaterName[0].
            equals("TOP")) {
         startDokument = i;
         //System.out.println("Start bei: "+startDokument);
      }
   }


   // erzeugen der logischen Struktur Vater - Sohn
   // fuer jedes gefundene Dokument
   for(int i = 0; i < gefundeneDokumente.size(); i++) {

      // fuer jeden Vater dieses Dokuments
      for(int j = 0;
          j < ((VollDokument)gefundeneDokumente.elementAt(i)).vaterName.length;
          j++) {

         // fuer alle gefundenen Dokumente
         for(int k = 0; k < gefundeneDokumente.size(); k++) {

            // welches Dokument ist der Vater
            if (((VollDokument)gefundeneDokumente.elementAt(i)).vaterName[j].
                  equals(((VollDokument)gefundeneDokumente.elementAt(k)).name)) {

               // Beim Sohn k-tes Dokument als Vater eintragen
               hilfsliste = bearbeiteteDokumente[i].vater;
               bearbeiteteDokumente[i].vater = new int[hilfsliste.length +1];
               for(int l = 0; l < hilfsliste.length; l++) {
                  bearbeiteteDokumente[i].vater[l] = hilfsliste[l];
               }
               bearbeiteteDokumente[i].vater[hilfsliste.length] = k;

               // Beim Vater i-tes Dokument als Sohn eintragen
               hilfsliste = bearbeiteteDokumente[k].sohn;
               bearbeiteteDokumente[k].sohn = new int[hilfsliste.length +1];
               for(int l = 0; l < hilfsliste.length; l++) {
                  bearbeiteteDokumente[k].sohn[l] = hilfsliste[l];
               }
               bearbeiteteDokumente[k].sohn[hilfsliste.length] = i;

            }
         }
```

```
      }
   }

   // Rueckgabe der kompletten Liste
   return bearbeiteteDokumente;
}

void Display() {

   // Variable zur verschiebung der Koordinaten
   Point Koo;

   // jedes Dokument als unbesucht markieren
   clearVisited();

   //berechnen der Koordinaten aller Dokumente
   // zuerst das Startdokument
   dokumentListe[startDokument].setzeKoordinaten(new Point(0,0));
   dokumentListe[startDokument].setVisited();

   // dann rekursive Berechnung der restlichen Struktur
   int kill = berechnePosition(0, 2*Math.PI, startDokument, addRadius);

   //Berechnen der endgueltigen Koordinaten
   int sizeHaelfteFlaeche = maxRadius;
   for (int i = 0; i < dokumentListe.length; i++) {
      Koo = new Point(0,0);
      Koo = dokumentListe[i].liesKoordinaten();
      Koo.x += sizeHaelfteFlaeche;
      Koo.y += sizeHaelfteFlaeche;
      dokumentListe[i].setzeKoordinaten(Koo);
   }
}


int berechnePosition(double from, double to, int dokumentNr, int radius) {

   int anzahlSoehne;
   double sektor;
   double winkel;
   boolean[] hierZuBesuchen;



   //Maximalen Wert fuer Radius feststellen
   if (maxRadius < radius)   maxRadius = radius;


   //Anzahl und Liste der noch nicht berechneten Soehne ermitteln

   // Anzahl der Soehne feststellen
   anzahlSoehne = dokumentListe[dokumentNr].sohn.length;
   // und Liste entsprechender Groesse erzeugen
```

```
hierZuBesuchen = new boolean[anzahlSoehne];

//fuer alle Soehne des aktuellen Dokuments
for (int i = 0; i < dokumentListe[dokumentNr].sohn.length; i++) {

   // fuer jeden bereits dargestellten Sohn ein Sektor weniger
   if (dokumentListe[dokumentListe[dokumentNr].sohn[i]].isVisited()) {
      anzahlSoehne = --anzahlSoehne;

      // und er muss nicht von dieser Inkarnation aus berechnet werden
      hierZuBesuchen[i] = false;
   }
   else {
      // ansonsten schon
      hierZuBesuchen[i] = true;
   }
}


//keine unbesuchten Soehne, keine Arbeit mehr :-)
if (anzahlSoehne == 0) return anzahlSoehne;



//Winkelbereich, der jedem Sohn zur Darstellung zur Verfuegung steht
sektor = (to - from)/anzahlSoehne;


//Ausrechnen der Koordinaten fuer alle noch nicht berechneten Soehne
// winkel initialisieren
winkel = (from + (sektor/2));
for (int i = 0; i < dokumentListe[dokumentNr].sohn.length; i++) {

   // wenn i-ter Sohn noch zu berechnen
   if (hierZuBesuchen[i]) {

      // Koordinaten des i-ten Sohnes berechnen
      dokumentListe[dokumentListe[dokumentNr].sohn[i]].setzeKoordinaten(
                           new Point((int)(radius * Math.cos(winkel)),
                           (int)(radius * Math.sin(winkel))));

      // Berechnungswinkel weiterschalten
      winkel = winkel + sektor;

      // und diesen Sohn als besucht markieren
      dokumentListe[dokumentListe[dokumentNr].sohn[i]].setVisited();
   }
}


//Erweitern der Darstellungsflaeche durch erhoehen des radius'
radius += addRadius;
```

```
      //rekursiver Aufruf von berechnePosition fuer alle Soehne
      double sektorende = sektor;
      for (int i = 0; i < dokumentListe[dokumentNr].sohn.length; i++) {

         // wenn i-ter Sohn noch unbesucht
         if (hierZuBesuchen[i]) {

            // rekursiver Aufruf mit diesem Sohn
            int Neffen = berechnePosition(from,
                                          from + sektorende,
                                          dokumentListe[dokumentNr].sohn[i],
                                          radius);

            // Darstellungsbereich nur weiterschalten, wenn Vorgaenger Soehne hatte
            if (Neffen != 0) {
               from = from + sektorende;
               sektorende = sektor;
            }
            else sektorende = sektorende + sektor;

            if (sektorende > MaxWinkel) {
               from = from + (sektorende - MaxWinkel);
               sektorende = MaxWinkel;
            }

         }
      }

      return anzahlSoehne;

   }

}
```

Literaturverzeichnis

[GM95] James Gosling and Henry McGilton. *The Java Language Environment - A White Paper*. Sun Microsystems Computer Company, USA, 1995.

[Inc96] Sun Microsystems Inc. *Java: Programming for the Internet*. http://www.javasoft.com/, 1996.

[LP96] Laura Lemay and Charles L. Perkins. *teach yourself JAVA in 21 days*. Sams.net Publishing, 1996.

[Sch96] Frank Schütz. *Implementierung eines Werkzeuges zur Konsistenzprüfung von HTML-Links*. Fortgeschrittenenpraktikum, Institut für Informatik der Technischen Universität München, 1996.