


# Web Loupe



**Entwicklerdokumentation**

Version 0.5

Februar 2005

*Ein Crawler  
zur Analyse und Visualisierung  
von Website-Strukturen*

**Ansprechpartner**

**Katja Langholz & Matthias Kahlau**

**Email:**

[kaimka@users.sourceforge.net](mailto:kaimka@users.sourceforge.net)  
[webspirit@users.sourceforge.net](mailto:webspirit@users.sourceforge.net)

**Webseiten:**

<http://www.webloupe.de.vu>  
<http://www.sourceforge.net/projects/webloupe/>

## Inhaltsverzeichnis

1 Einleitung.....	4
1.1 Was ist WebLoupe.....	4
1.2 Lizenz.....	4
1.3 Möglichkeiten zur Mitwirkung.....	4
1.4 Entwicklerdokumentation.....	4
2 Allgemeines .....	5
2.1 Programmiersprache.....	5
2.2 Software dritter Parteien.....	6
2.3 Systemvoraussetzungen.....	6
2.3.1 Zur Ausführung von WebLoupe.....	6
2.3.2 Zur Entwicklung von WebLoupe.....	7
2.4 Entwicklungsumgebung.....	7
3 Die Softwarekomponenten.....	7
3.1 Crawler.....	7
3.2 Visualisierung.....	11
3.2.1 Baumstruktur mit einem JTree.....	11
3.2.2 Tabellenansicht.....	13
3.2.3 TouchGraph.....	14
3.2.4 HyperGraph.....	16
3.3 Standard GUI .....	17
3.4 Properties und I/O.....	19
3.5 Utils.....	21
4 Ausblick.....	22
5 Quellenangaben.....	23

# 1 Einleitung

## 1.1 Was ist WebLoupe

WebLoupe ist eine Software zur Analyse und Visualisierung von Webseiten und ihren Verknüpfungen. Sie basiert auf einer Web Crawler-Technologie (auch Spider oder Robot genannt), d.h. Webseiten werden nach Links durchsucht und diese weiterverfolgt, um weitere Webseiten einzubeziehen. Dabei können Daten der einzelnen Webseiten gesammelt werden, wie Titel, Dateigröße, enthaltene Bilder etc.. Was genau ein Crawler leisten kann, hängt von der Charakteristik der einzelnen Lösungen ab. WebLoupe ist nicht der einzige Web Crawler, den es gibt, soll aber eine echte Alternative zu anderen Produkten darstellen.

Die Entwicklung von WebLoupe hat im November 2004 im Rahmen eines Studienprojektes der Informatik an der Fachhochschule Kaiserslautern, Standort Zweibrücken begonnen. Es sollte eine Software zum Thema Informationsarchitektur entwickelt werden. Dabei mussten Risiken wie Visualisierung und Performanz der Anwendung, sowie die relativ knappe Entwicklungszeit zum Abgabetermin im Februar 2005 berücksichtigt werden. Die genaue Ausprägung der Software wurde aber vom Dozenten nicht festgelegt, sondern konnte nach eigenen Vorstellungen entwickelt werden. Vorgabe war jedoch die Entwicklung unter einer Open-Source Lizenz. Das bedeutet, dass WebLoupe nach Abgabe der ersten Version von den ursprünglichen Entwicklern und von Interessenten weiter entwickelt werden kann, z.B. zur Implementation zusätzlicher Features.

## 1.2 Lizenz

WebLoupe wird unter der Open-Source Lizenz GNU General Public License (GPL) entwickelt. Mehr Informationen hierzu unter <http://www.opensource.org>. Die Lizenz finden sie ebenfalls auf der Produktwebseite von WebLoupe unter <http://www.webloupe.de.vu>.

## 1.3 Möglichkeiten zur Mitwirkung

WebLoupe sollte in erster Linie als Kernversion zum Abgabetermin fertig gestellt sein, d.h. die essentiellen Features sollten bis dahin vorhanden sein. Dennoch kann WebLoupe natürlich darüber hinaus weiterentwickelt werden, und zwar nicht nur von den ursprünglichen Entwicklern, sondern auch von anderen Interessierten, die an einem Open-Source Projekt mitwirken möchten.

Eine gute Möglichkeit zur gemeinsamen Softwareentwicklung und Kommunikation an verschiedenen Orten wird von der größten Open-Source Plattform des Internets – Sourceforge.net – angeboten. Deshalb wird WebLoupe als Sourceforge-Projekt entwickelt. Unter der Adresse <http://sourceforge.net/projects/webloupe/> können neben dem Download der Dateien von WebLoupe zahlreiche andere Dienste wie Foren, BUG Report und weitere Kommunikationsmittel genutzt werden.

## 1.4 Entwicklerdokumentation

Diese Dokumentation wurde mit dem Ziel geschrieben, Entwickler bei der Programmierung von WebLoupe oder bei der Verwendung von Teilen des Programmcodes zu unterstützen. Dies ist allerdings keine API Dokumentation. Diese ist im üblichen HTML-Format auf der Projektwebseite

<http://www.webloupe.de.vu> erhältlich, bzw. im Entwicklerpaket von WebLoupe enthalten, dem sie vielleicht diese Dokumentation entnommen haben.

Für das Verständnis dieser Dokumentation werden grundlegende Kenntnisse des Java 5 API vorausgesetzt.

## 2 Allgemeines

### 2.1 Programmiersprache

WebLoupe wurde mit der Programmiersprache Java unter Verwendung der Java 2 Standard Edition (J2SE) 5.0 der Firma SUN Microsystems realisiert. Mit dieser erst vor kurzem veröffentlichten Version sind einige Verbesserungen vorgenommen worden und Erweiterungen hinzugekommen.

SUN hat sich dabei auf folgende Schlüsselthemen konzentriert [1]:

- Ease of Development
- Skalierbarkeit und Performanz (Scalability and Performance)
- Überwachung und Handlichkeit (Monitoring and Manageability)
- Desktop Client

Die Programmierung mit dem neuen Java 5.0 API (Application Programming Interface) kann sowohl Programmierern wie auch Endanwendern einen Vorteil gegenüber älteren Versionen bringen:

- schnellere Startzeit der Anwendung,
- höhere Performanz zur Laufzeit durch einen integrierten automatischen Optimierer,
- weniger Speicherverbrauch durch eine intelligenteren Verwaltung,
- moderneres Look & Feel,
- bessere Unterstützung für die Programmierung von multithreaded Anwendungen

Eine sehr wichtige Eigenschaft von Java ist die Plattformunabhängigkeit, die SUN's Slogan „Write once, run anywhere!“ begründet hat. Mit Java Swing kann die resultierende Anwendung problemlos für die gängigsten Betriebssysteme im jeweiligen nativen Look & Feel zur Verfügung gestellt werden, wodurch Benutzer das System schneller lernen können. Die J2SE 5.0 bringt aber auch ein neues und frisches Cross-Plattform Look & Feel mit, das „Ocean“ genannt wurde. Zusätzliche Pakete ermöglichen sogar skinnable Look & Feels.

Neben dem Look & Feel bedeutet aber Plattformunabhängigkeit vor allem, dass Java Anwendungen ohne plattform-spezifische Programmierung und Kompilierung auf allen populären Plattformen ausgeführt werden können, da der kompilierte und plattformunabhängige Java-Bytecode von einer virtuellen Maschine ausgeführt wird.

Ein weiterer Faktor für die Entscheidung für Java ist die weite Verbreitung der Programmiersprache. Es gibt viele Open-Source Programme und Bibliotheken, die mit Java entwickelt wurden.

Daher kann Software Dritter leicht in den Code von WebLoupe eingebunden werden. So muss das Rad nicht ständig neu erfunden werden, und die Entwicklungszeit wird verkürzt.

## **2.2 Software dritter Parteien**

Für WebLoupe wurde bisher folgende freie Software integriert:

- TouchGraph (<http://www.touchgraph.com>) zur interaktiven Visualisierung von Informationsarchitekturen (Website-Strukturen)
- BrowserLauncher (<http://browserlauncher.sourceforge.net/>) zum Öffnen von URL's im Standard-Webbrowser eines Betriebssystems

## **2.3 Systemvoraussetzungen**

### **2.3.1 Zur Ausführung von WebLoupe**

Voraussetzung für die Ausführung von WebLoupe ist ein passendes Java Runtime Environment (JRE) der Java 2 Standard Edition (J2SE). Für WebLoupe wird mindestens die aktuelle Version 5.0 (auch 1.5 genannt) benötigt. Zu älteren Versionen ist der Sourcecode und Bytecode von WebLoupe aufgrund der Compatibility Policies von SUN nicht kompatibel.

Diese Version von Java ist auf den aktuellen Betriebssystemen nicht vorinstalliert, d.h wenn sie dort noch nicht zur Verfügung steht, muss sie für WebLoupe bereitgestellt werden.

Das JRE 5.0 kann kostenlos unter folgender Adresse heruntergeladen werden (ca. 15 MB):  
<http://java.sun.com/j2se/1.5.0/download.jsp>

Allerdings wird WebLoupe nicht nur in Form eines JARs mit den Java-Klassen angeboten, für die ein JRE installiert sein muss, sondern auch als Windows .exe Datei.

Die Windows .exe gibt es in 2 Varianten, einmal ohne und einmal mit JRE:

Die Variante ohne JRE sucht nach Doppelklick nach der benötigten JRE. Ist keine JRE 5.0 auf dem Zielsystem vorhanden, wird der Anwender über einen Dialog benachrichtigt und kann die JRE nach entsprechender Auswahl direkt von oben genannter Webseite des Herstellers herunterladen. Die zweite Variante ist zwar entsprechend größer, dafür wird aber keine vorherige Installation der JRE benötigt. Sie ist im Programmpaket von WebLoupe enthalten, d.h. WebLoupe kann nach Doppelklick auf die .exe sofort gestartet werden, unabhängig von dem Vorhandensein einer JRE auf dem Zielsystem.

Die mit WebLoupe gebündelte JRE ist unsichtbar für das Betriebssystem, da sie nicht installiert wird. Sie wird daher nicht automatisch von anderen Anwendungen verwendet.

## 2.3.2 Zur Entwicklung von WebLoupe

Zur Entwicklung muss das Java Development Kit (JDK) 5.0 der Java 2 Standard Edition (J2SE) verwendet werden.

Es kann kostenlos unter folgender Adresse heruntergeladen werden (ca. 50 MB):

<http://java.sun.com/j2se/1.5.0/download.jsp>

## 2.4 Entwicklungsumgebung

WebLoupe wurde mit dem professionellen und kostenlos erhältlichen Eclipse Software Development Kit (SDK) 3.1 M4 entwickelt, das die neuen Sprachfeatures von Java 5.0 wie Enumerationen, Enhanced for-Loop etc. unterstützt. Dabei handelt es sich um ein sog. Stream Stable Build, kein Release. Das letzte Eclipse Release 3.0.1 unterstützt noch nicht alle neuen Java Features.

Die Software umfasst viele nützliche Features zur Unterstützung eines optimalen Entwicklungsprozesses. Mit dem CVS Plugin von Eclipse lassen sich beispielsweise direkt von Sourceforge.net der Quellcode und weitere benötigte Dateien von WebLoupe herunterladen und als Projekt anlegen. Voraussetzung dafür ist allerdings, dass man einen Entwickleraccount für WebLoupe hat. Interessierten Entwicklern kann so ein Account von uns auf Sourceforge.net bereitgestellt werden.

Natürlich kann man den Quellcode von WebLoupe auch ohne Entwickleraccount von der Projektwebseite herunterladen, allerdings nicht via CVS und dann ist auch keine direkte Mitwirkung am Projekt via Sourceforge.net möglich.

Das Eclipse SDK kann unter folgender Adresse kostenlos heruntergeladen werden:

<http://www.eclipse.org/downloads/index.php>

## 3 Die Softwarekomponenten

### 3.1 Crawler

Die Klassen für den Crawler sind im Paket `de.webloupe.crawler` enthalten. Es hat folgenden Inhalt:

#### Interfaces:

- ICrawler

#### Klassen:

- BreadthCrawlerMT
- Settings

#### Enums:

- `CrawlingState`
- `MimeType`
- `Scope`

Der Algorithmus für das Crawlen von Webseiten wurde als Breitensuche implementiert. Die Struktur, die Webseiten durch ihre Verknüpfungen bilden, kann man sich als eine Baumstruktur bzw. als Graph mit Knoten und Kanten vorstellen.

Eine Webseite enthält meist eingebettete URL's, die auf weitere Seiten verweisen. Geht man von der Startseite aus, wären das die Kinder in der ersten Ebene. Diese Kinder haben wiederum eingebettete Links, die auf weitere Seiten verweisen. Das sind die Kinder der zweiten Ebene, u.s.w..

Bei einer Breitensuche werden erst alle Webseiten einer Ebene gesammelt, dann die der nächsten Ebene u.s.w.. Die Tiefensuche würde im Gegensatz erst so tief wie möglich von einer Webseite zu nächsten gehen, und erst dann einen weiteren Link auf der Startseite verfolgen, wieder so tief wie möglich bzw. wie durch mögliche Parameter eingestellt.

Bei der Breitensuche von WebLoupe wird jede URL nur einmal berücksichtigt, d.h. wird eine URL mehrmals gefunden, wird sie nach dem ersten Mal nur noch verworfen. Damit sollen zum einen Endlosschleifen des Crawlers vermieden werden, und zum anderen die Komplexität der Strukturen reduziert werden, um sie besser visualisieren zu können.

Das Interface `ICrawler` definiert grundlegende Methoden wie `start()` und `stop()` zur Steuerung des Crawlers und `getStartUrl()` um die URL des Crawlers abzufragen, mit der er gestartet wurde. Das Interface wird dort als Referenz eingesetzt, wo nur Basisfunktionen benötigt werden. Wird zum Beispiel ein weiterer Crawler für WebLoupe implementiert, kann das selbe Interface für diese Funktionen verwendet werden.

Die Klasse `BreadthCrawlerMT` implementiert sämtliche Funktionen, die für den Crawlvorgang benötigt werden, greift aber auf die Utility-Klasse `UrlUtil` im Paket `de.webloupe.utils` zurück. Die Aufgaben der `UrlUtil` Klasse werden im Kapitel 3.5 erläutert.

Die Klasse `BreadthCrawlerMT` hat drei innere Klassen, die auf verschiedene Member der äußeren Klasse zugreifen, um Variablen und Klassen zu lesen oder Werte von Variablen zu verändern:

- `DownloadTask`
- `ParserTask`
- `SpiderParserCallback`

Die Klasse `DownloadTask` hat die Aufgabe, eine URL herunterzuladen. Wenn dies erfolgreich ist, dann übergibt er die heruntergeladenen Daten in Form eines `InputStreams` bzw. `InputStreamReaders` an einen `ParserTask`. Dieser hat dann die Aufgabe, die Ressource nach Titel, eingebetteten URL's etc. zu parsen.

`DownloadTask` und `ParserTask` implementieren beide das Interface `java.lang.Runnable` und werden als eigenständige Threads ausgeführt. Da es beim Downloaden zu Verzögerungen bzw. Wartezeiten kommen kann, werden mehrere `DownloadTasks` parallel ausgeführt, um die Performanz des Crawlvorgangs zu steigern. Dazu werden neue Klassen des

`java.util.concurrent` Paketes eingesetzt, die mit dem JDK 5.0 eingeführt wurden.

Für die `DownloadTasks` wird ein Thread Pool fixer Thread-Anzahl, momentan in der Größe von max. 15 Threads, verwendet. Dieser „fixed“ Thread Pool kann mit der Factory-Klasse `java.util.concurrent.Executors` erzeugt werden.

`Executors` stellt auch einen Thread Pool mit sich dynamisch anpassender Größe bereit, der mit der Anzahl der Aufgaben beliebig nach oben skaliert. Dieser hat sich aber als ungeeignet herausgestellt, da je nach Anzahl der Seiten einige Hundert Threads erzeugt werden – einfach zuviel.

Dem mit `Executors` erzeugten `java.util.concurrent.ThreadPoolExecutor` können dann `Runnable` Objekte als Tasks übergeben werden, die in eine Queue gestellt werden und von den Threads des Pools parallel abgearbeitet werden. Dieses `Runnable`s sind unsere `DownloadTasks`.

Die `DownloadTasks` werden in einer while Schleife der `execute(Runnable)` Methode des `ThreadPoolExecutor` übergeben. Die while Schleife befindet sich in der `run()` Methode der Klasse `BreadthCrawlerMT`. Sie wird ausgeführt, wenn der Crawler mit der `start()` Methode gestartet wird und terminiert, wenn das `isRunning` Flag in der `stop()` Methode des Crawlers auf `false` gesetzt wird. Die `start()` und `stop()` Methoden werden ausgeführt, wenn der Anwender den Crawler startet oder stoppt.

Dem Konstruktor eines `DownloadTask` wird ein `javax.swing.DefaultMutableTreeNode` übergeben. Dieser Knoten kann eine Referenz zu einem User Objekt enthalten, das ein beliebiges Objekt sein kann. Ich habe dafür die Klasse `de.webloupe.tree.UserObject` implementiert. Sie kapselt sämtliche gewonnen Informationen zu einer URL, z.B. Titel, Dateigröße, Http Response Code etc.. Dadurch kann der `DownloadTask` auf die URL zugreifen und Attribute setzen.

Es wird deswegen ein `DefaultMutableTreeNode` verwendet, weil diese Knoten zu einer Baumstruktur verknüpft werden können, und auch in einem `javax.swing.JTree` angezeigt werden können.

Die Knoten werden im Parser erzeugt, wenn eingebettete URL's gefunden werden. Diese Knoten werden dann in eine FIFO Queue gestellt, die in der oben erwähnten while-Schleife der `run()` Methode der Klasse `BreadthCrawlerMT` abgearbeitet wird, indem die einzelnen Knoten einem `DownloadTask` übergeben werden.

Für die Queue wird eine thread-sichere `java.util.concurrent.LinkedBlockingQueue` verwendet. Die Schleife terminiert nicht nur durch explizites Stoppen des Anwenders, sondern auch, wenn die Queue leer ist und keine `DownloadTasks` oder `ParserTasks` mehr aktiv sind bzw. bearbeitet werden müssen

Für `ParserTasks` wird ebenfalls ein `ThreadPoolExecutor` verwendet, allerdings nur mit einer Größe von einem Thread, da beim Parsen keine Wartezeiten auftreten und so alle `ParserTasks` sequentiell abgearbeitet werden können.

Zum Parsen wird ein `javax.swing.text.html.parser.ParserDelegator` verwendet, dessen `parse(...)` Methode neben einem `InputStreamReader` ein `HTMLToolkit.ParserCallback` übergeben werden muss. Dazu habe ich die Klasse `SpiderParserCallback` als innere Klasse von `BreadthCrawlerMT` implementiert, die von `HTMLToolkit.ParserCallback` abgeleitet ist.

Sie enthält verschiedene überschriebene Methoden der Superklasse, die aufgerufen werden, wenn der Parser auf HTML Elemente bzw. Inhalte stößt, z.B. Starttags, Endtags, Text. In den Methoden können dann die jeweiligen Elemente bzw. Inhalte analysiert und weiterverarbeitet werden. So kann z.B. der Titel einer Seite gefunden werden, oder die eingebetteten URL's, mit denen ein neues `UserObject` erzeugt wird.

Mit diesem wird dann ein `DefaultMutableTreeNode` erzeugt, der dem Elternknoten, der gerade geparkt wird, als Kindknoten hinzugefügt wird, und in die FIFO Queue zum Download und anschliessendem Parsen gestellt wird.

Im `ParserTask` werden also die Knoten anhand gefundener URL's erzeugt, im `DownloadTask` wird dann versucht, die URL herunterzuladen und anschliessend einem `ParserTask` zu parsen, wobei jeweils wichtige Daten im `UserObject` des Knotens gespeichert werden können.

Daten, die pro Knoten (Webseite) in einem `UserObject` gespeichert werden sind:

- URL
- Base URL
- Titel
- HTTP Response Code
- Dateigröße
- Mime Typ
- Anzahl gefundener Zeichen auf einer Webseite
- Anzahl gefundener Bilder auf einer Webseite
- Vorhandensein von Keywords auf einer Webseite
- Der Zustand bzw. das Ergebnis des Crawlens der Webseite (`CrawlingState`)
- Elternknoten (Knoten der URL, die zuerst auf diese URL verwiesen hat)

Die Knoten werden schon während des Crawlens in einem `JTree` angezeigt. So können gefundene Seiten sofort gesehen und der Fortschritt des Crawlvorgangs verfolgt werden.

Weitere Informationen dazu im Kapitel 3.2.1 (`JTree`).

Nach dem Crawlen, entweder wenn der Crawler den Suchvorgang selbst beendet, oder nachdem der Benutzer den Crawler gestoppt hat, wird der komplette `JTree` angezeigt. Darunter wird außerdem eine Tabelle angezeigt, die in jeder Zeile verschiedene Informationen einer Webseite anzeigt. Welche Spalten angezeigt werden, kann vom Benutzer eingestellt werden. Weitere Informationen zur Tabellenansicht im Kapitel 3.2.2.

Für den Crawler können eine Menge von Einstellungen durch den Benutzer vorgenommen werden.

Sie werden in einem `de.webloupe.crawler.Settings` Objekt gespeichert. Es sind folgende Attribute per GUI einstellbar:

- max. Anzahl der Webseiten die gesammelt werden sollen; ist die Anzahl erreicht, bricht der Crawlvorgang ab
- max. Tiefe der Ebenen; ist die Tiefe erreicht, bricht der Crawlvorgang ab
- Einschränkung des Suchbereichs (Enum `de.webloupe.crawler.Scope`):
  - alle Seiten
  - nur innerhalb der Startseite
  - nur .de Domäne
  - nur .com Domäne
  - nur .org Domäne
  - nur .net Domäne
  - nur .edu Domäne

Die Startseite bleibt von der Beschränkung ausgeschlossen.

- Einschränkung des Mime Typs (Enum `de.webloupe.crawler.MimeType`)
  - Alle
  - nur text/html
- max. Zeit, nach der ein Verbindungsversuch zu einem Server abgebrochen wird
- max. Dateigröße von Ressourcen; größere Dateien werden erst gar nicht heruntergeladen, bzw. vor dem Parsen verworfen
- außerdem können Keywords angegeben werden, nach denen auf den Webseiten gesucht werden soll

Der `CrawlingState` bezeichnet den Zustand einer Resource, der sich im Laufe des Verarbeitungsprozesses ändern kann. In der Enum wurden folgende Zustände definiert:

- `Discovered` – der Anfangszustand, nachdem die Resource gefunden wurde und als `UserObject` existiert
- `Fetches` – nachdem die Resource erfolgreich heruntergeladen wurde
- `Parsed` – nachdem die Resource erfolgreich geparkt wurde
- `FetchError` – die Resource konnte nicht erfolgreich heruntergeladen werden (z.B. wenn `Http Response Code < 200` oder `> 302`)
- `ParseError` – die Resource konnte nicht erfolgreich geparkt werden; es ist eine Exception während des Parsens aufgetreten
- `MalformedURLException` – die URL Syntax ist inkorrekt
- `ContentLengthOversized` – die max. Dateigröße ist überschritten
- `MimeTypeExcluded` – der Mime Typ der Datei ist ausgeschlossen
- `ContentLengthOversized_MimeTypeExcluded` – Kombination aus den einzelnen Zuständen
- `Timeout` – die max. Wartezeit für den Verbindungsaufbau ist überschritten

## 3.2 Visualisierung

### 3.2.1 Baumstruktur mit einem JTree

Die Klassen für die Visualisierung mit einem `JTree` sind abgesehen vom

`de.webloupe.gui.TreeViewPanel`, auf dem der `JTree` dargestellt wird, im Paket `de.webloupe.tree` enthalten. Es hat folgenden Inhalt:

### Klassen:

- `CustomTreeCellRenderer`
- `CustomTreeListener`
- `UserObject`

Wie schon im Kapitel über den Crawler erwähnt, werden für jede gefundene URL Ressource Knoten erzeugt, die zu einer Baumstruktur zusammengefügt werden. Dafür bietet die Swing API gute Unterstützung. Die Knoten können mit einem `DefaultMutableTreeNode` des Pakets `javax.swing.tree` erzeugt werden. Jeder Knoten kann ein sog. User Object referenzieren, das ein beliebiges Objekt sein kann. Für dieses Objekt wurde die Klasse `UserObject` (im Paket `de.webloupe.tree`) implementiert. In ihm werden sämtliche gewonnenen Daten zu einer Webseite gespeichert.

Die Datenstruktur ist theoretisch unabhängig von der Ansicht, wird aber schon während des Crawlens in einem `JTree` angezeigt. Dazu wird der `JTree` bereits zu Beginn, bevor der Crawler startet, einer eigenen Subklasse von `JPanel` zugewiesen, dem `TreeViewPanel`, auf dem der `JTree` gerendert wird.

Bei der Erzeugung des `JTree` wird dem Konstruktor der Wurzelknoten übergeben, der die Startseite repräsentiert. Alle weiteren Knoten werden nach dem Finden eingebetteter URL's im Parser erzeugt, und dem `TreeModel` des Baumes hinzugefügt, so dass der `JTree` ständig aktualisiert wird.

Zu Beginn kann allerdings nur die URL und kein Titel eines Knotens angezeigt werden, da eine gefundene URL ja bei der Breitensuche erst später geparkt wird, und der Knoten solange in die FIFO Warteschlange gestellt wird.

Zum Rendern des Baumes wurde der `de.webloupe.tree.CustomTreeCellRenderer` geschrieben. Das war notwendig, um Broken Links in roter Farbe hervorheben zu können. Die Klasse ist von `JLabel` abgeleitet und implementiert das `TreeCellRenderer` Interface. In der Klasse existiert nur eine Methode, die `getTreeCellRendererComponent(...)`, die für jeden Knoten aufgerufen wird. In dieser Methode können Eigenschaften des Knotens abgefragt werden und Attribute der `Component`, die zurückgegeben wird, entsprechend gesetzt werden.

Bei der Auswahl eines Knotens (bei uns kann immer nur einer ausgewählt werden, wegen des eingestellten Modus `SINGLE_TREE_SELECTION`) wird die korrespondierende Zeile in der darunter liegenden Tabelle angezeigt. Die Tabelle zeigt pro Zeile verschiedene Daten einer Ressource. Dies geht allerdings erst, wenn der Crawlvorgang beendet ist, da die Tabelle erst danach dargestellt wird.

Um diese Auswahl zu realisieren habe ich den `de.webloupe.tree.CustomTreeListener` geschrieben. Sie erbt von `java.awt.event.MouseAdapter` und implementiert `javax.swing.event.TreeSelectionListener`.

Zur Auswahl einer Tabellenzeile wird nach Auswahl eines Knotens die Methode `valueChanged (TreeSelectionEvent)` des Interface `TreeSelectionListener` aufgerufen. Dort wird die URL des Knotens ausfindig gemacht, und dann die Tabelle nach der Zeile mit der URL durchsucht, welche dann ebenfalls ausgewählt, und evt. sichtbar gescrollt wird. Dies geht allerdings im Moment nur, wenn als Spalte in der Tabelle auch die URL's angezeigt werden. Dies muss aber nicht immer der Fall sein, da es optional ist, welche Spalten angezeigt werden.

Die Superklasse `MouseAdapter` wird benötigt, um auf Knoten-Doppelklicks reagieren zu können. Dann wird der Standardbrowser mit der entsprechenden URL geöffnet. Daher wurde die Anzahl der benötigten Klicks zum Aufklappen oder Einklappen der Kinder von Knoten auf Drei erhöht.

Alle Einstellungen des `JTree` werden in der Methode `initTree()` der Klasse `de.webloupe.crawler.BreadthCrawlerMT` vorgenommen.

Zum Öffnen der URL im Standardbrowser wird die Klasse `BrowserLauncher` des Pakets `de.webloupe.utils` verwendet. Weitere Informationen zu dieser Klasse finden sie im Kapitel 3.5.

### 3.2.2 Tabellenansicht

Für die Funktionalität der Tabelle existiert kein eigenes Paket. Sie wurde innerhalb der Klasse `de.webloupe.gui.StatisticsPanel` untergebracht, das `JPanel`, auf dem die Tabelle dargestellt wird. Für das `TableModel` habe ich eine innere Klasse geschrieben, `StatisticsTableModel`, die von `javax.swing.table.AbstractTableModel` abgeleitet ist.

Es wurden folgende Methoden überschrieben und entsprechend ihren Forderungen implementiert:

- `public int getColumnCount()`
- `public int getRowCount()`
- `public Class<?> getColumnClass(int columnIndex)`
- `public String getColumnName(int columnIndex)`
- `public Object getValueAt(int rowIndex, int columnIndex)`

Für die Spaltennamen wird eine `ArrayList<String>` verwendet. Die Spaltennamen sind in der privaten Enum `StatisticsColumnName` der Klasse `StatisticsPanel` definiert.

Für die Daten pro Zeile wird eine `ArrayList<ArrayList>` verwendet. Die `ArrayList` enthält also `ArrayListen`, die die Werte der Spalten einer Zeile enthält, in der gleichen Reihenfolge wie die Spaltennamen.

Die Tabelle wird nach Ende des Crawlvorgangs in einem `JPanel` unterhalb der `JTree` Ansicht mit den Ergebnissen angezeigt. In jeder Zeile wird ein Knoten des Baumes angezeigt, als eine zu einer URL gehörige Ressource. Welche Spalten angezeigt werden, hängt von den Benutzereinstellungen ab.

Folgende Spalten können angezeigt werden:

- URL
- Titel
- HTTP Response Code
- Status (Discovered, Fetched, Parsed, FetchError, ParseError, MalformedURLException, ContentLengthOversized, MimeTypeExcluded, ContentLengthOversized\_MimeTypeExcluded, Timeout) – siehe Kapitel 3.1.
- Mime Type
- Anzahl der Zeichen pro Seite
- Anzahl der Bilder pro Zeichen
- Dateigröße
- gefundene Keywords pro Seite

Nach dem Crawlvorgang wird das `TableModel` mit dem Aufruf von `StatisticsTableModel.initTable(TreeModel)` konfiguriert und mit Daten gefüllt. Im `TreeModel` ist wieder die durch den Parser erstellte Baumstruktur enthalten.

Abhängig von den Benutzereinstellungen, die in den `de.webloupe.crawler.Settings` gespeichert sind, wird dann die `ArrayList<String>` mit den gewünschten Spaltennamen erzeugt. Dann wird über das `TreeModel` in einer Breadth-First Enumeration iteriert, um die `ArrayList<ArrayList>` von Zeilen mit den `ArrayList`en von Spaltenwerten zu erzeugen. Dazu werden die Daten von den jeweiligen `de.webloupe.tree.UserObjects` abgerufen, die zu den Knoten des Baumes gespeichert sind.

Wählt man eine Zeile der Tabelle aus, wird nicht der korrespondierende Knoten im `JTree` ausgewählt. Das geht bisher nur umgekehrt. Beim Doppelklick auf eine Zeile der Tabelle öffnet sich auch kein Webbrowser mit der URL, wie im `JTree`. Dies Funktionen wären aber denkbar für zukünftige Versionen. Bei der Auswahl muss man allerdings darauf achten, dass sich kein Ping-Pong zwischen den Aufrufen der Listener-Methoden von `JTree` und `JTable` ergibt.

### 3.2.3 TouchGraph

Die Klassen für die Visualisierung des TouchGraph sind im Paket `de.webloupe.touchgraph` enthalten. Es hat folgenden Inhalt:

#### Interfaces:

- `TouchGraphNode`

#### Klassen:

- `TouchGraphInitializer`
- `Screenshot`

Die Klasse `TouchGraphInitializer` stellt die Schnittstelle von WebLoupe zu den wirklichen Klassen von TouchGraph dar. Diese Klassen sind in folgenden Paketen enthalten:

- `com.touchgraph.graphlayout`

- `com.touchgraph.graphlayout.graphelements`
- `com.touchgraph.graphlayout.interaction`

TouchGraph ist ein open-source Java Programm, das zur Visualisierung von Graphen, also Gebilden aus Knoten und Kanten dient. Ein Knoten kann beliebig viele Kinder haben. In den Knoten werden die Titel der Webseite angezeigt. Die Visualisierung ist interaktiv, d.h. Knoten können verschoben und die Baumansicht verändert werden.

Zu Beginn werden alle Knoten aller Ebenen angezeigt. Klickt man auf einen Knoten, wird er in die Mitte der Zeichenfläche verschoben, und alle Knoten außer seinen Kindern werden ausgeblendet.

Knoten, deren Kinder ausgeblendet sind kann man daran erkennen, dass eine kleine Ziffer für die Anzahl im Knoten erscheint. Klickt man auf einen solchen Knoten, wird dieser wiederum in die Mitte verschoben, seine Kinder angezeigt, und alle anderen Knoten ausgeblendet. Die Mausfunktionen können auch im Kontextmenü ausgewählt werden.

Neben den geschilderten Funktionen gibt es noch die Möglichkeit, den TouchGraph zu zoomen und zu rotieren, oder einen Linseneffekt ein- bzw. auszuschalten, der den TouchGraph in der Mitte vergrößert und nach außen hin verkleinert, indem die Kantenlänge verändert wird.

Der TouchGraph kann erst nach dem Crawlvorgang angezeigt werden. Dies geschieht dann automatisch, wenn man in der `JTabbedPane` der GUI das TouchGraph Tab auswählt.

Um den TouchGraph zu erzeugen, habe ich die Klasse `TouchGraphInitializer` geschrieben. Dort wird über das `TreeModel` mit den Knoten iteriert, die aus dem Crawlvorgang hervorgehen, um Knoten und Kanten des TouchGraph zu erstellen. Zur Iteration wird eine *Breadth-First Enumeration* verwendet, d. h. die Knoten einer Ebene kommen auf jeden Fall vor ihren Kindern, vom ersten Knoten aus gesehen.

Für jeden Baumknoten wird ein TouchGraph Knoten mit der Methode `addNode(String, String)` der Klasse `com.touchgraph.graphlayout.TGPanel` erzeugt. Der Methode werden die ID und der Titel eines `UserObjects` übergeben.

Für alle Knoten außer dem Wurzelknoten wird eine Kante zu seinem Elternknoten erzeugt. Dafür wird eine `HashMap` verwendet, in der jeder TouchGraph Knoten mit seiner unikaten ID, die im `UserObject` festgelegt ist, als Schlüssel gespeichert wird.

Von jedem `UserObject` kann der Elternknoten, und von dem das `UserObject` abgefragt werden. Vom Eltern-`UserObject` wird dann die ID abgefragt und der zugehörige TouchGraph Knoten aus der `HashMap` geholt. Mit diesen beiden Knoten kann dann die Methode `addEdge(Node, Node, int)`, ebenfalls aus der `TGPanel` Klasse, aufgerufen werden, um die Kante zu erzeugen. Der dritte Parameter ist die *Tension* oder die Länge einer Kante, und wurde nach Experimentieren mit verschiedenen Werten auf Zehn festgelegt.

Alles Weitere, wie das interaktive Verhalten, ist bereits in den TouchGraph Klassen implementiert. Denkbar wäre in zukünftigen Versionen von WebLoupe, dass nach Doppelklick auf einen Knoten,

wie bei dem `JTree`, die zugehörige Webseite mit dem Standardbrowser geöffnet wird, denn das wird momentan noch nicht unterstützt.

Damit `UserObjects` die Semantik für einen `TouchGraph` Knoten erfüllen, implementiert die Klasse `UserObject` das Interface `TouchGraphNode`, das Methoden festlegt, um die nötigen Informationen über einen Knoten bei der Erstellung eines `TouchGraph` zu bekommen.

Die Klasse `Screenshot` wurde geschrieben, um den erzeugten `TouchGraph` als Bild speichern zu können. Das Bild wird im PNG Format gespeichert. Die Erzeugung von Bildern ist möglich, sobald der `TouchGraph` erzeugt wurde, also nach Ende des Crawlvorgangs, auch wenn eine andere Ansicht als der `TouchGraph` in der GUI ausgewählt ist. Es wird nämlich nicht wirklich ein Bildschirmfoto gemacht, sondern ein Offscreen-Image (`java.awt.image.BufferedImage`) des `TouchGraph` erzeugt, indem die `paint(Graphics)` Methode des `TGPanel`s aufgerufen wird, auf dem der `TouchGraph` gerendert wird. Als Parameter für `Graphics` wird die Referenz auf das Offscreen-Image verwendet.

Weitere Informationen zu `TouchGraph` und Beispiele finden sie auf dessen Projektwebseite:

<http://www.touchgraph.com/>

### 3.2.4 HyperGraph

Die Klassen zur Visualisierung eines `HyperGraph` sind im Paket `de.webloupe.hypergraph` enthalten. Es hat folgenden Inhalt:

#### Interfaces:

- `GraphXMLNode`

#### Klassen:

- `GraphXMLBuildingBlocks`
- `GraphXMLFactory`
- `HyperGraphExporter`

`HyperGraph` ist ein open-source Java Programm bzw. eine Bibliothek zur Visualisierung von Graphen, also Knoten, die über Kanten verbunden sind. Ein Knoten kann beliebig viele Kinder haben.

Der Unterschied zu `TouchGraph` besteht in der Darstellung des Graphen bzw. im Visualisierungsalgorithmus. `HyperGraph` verwendet hyperbolische Geometrie, speziell hyperbolische Bäume um einen Linseneffekt zu erzeugen. `HyperGraph` ist ebenfalls interaktiv, hier wird allerdings nach Auswahl eines Knoten der Standardbrowser mit der URL geöffnet.

Die `HyperGraph` Visualisierung wurde nicht in `WebLoupe` integriert, da die Klassen beim Ausprobieren nicht zum gewünschten Ergebnis geführt haben und nur dürftig kommentiert sind. Eine Entwicklerdokumentation ist noch in Arbeit. Das `HyperGraph` Applet, das als `hyperapplet.jar` von den Entwicklern zur Verfügung gestellt wird, funktionierte allerdings

sofort.

In WebLoupe wurde daher eine Exportfunktion implementiert, welche die erstellte Baumstruktur als GraphXML in eine Datei schreiben kann, die dann vom HyperGraph Applet eingelesen werden kann. GraphXML ist eine Beschreibungssprache für Graphen in XML, die Attribute von Knoten und deren Verknüpfungen ausdrücken kann.

Zur Ausführung des Applets wird als dritte Datei noch die GraphXML.dtd benötigt. Diese Datei, und die Datei `hyperapplet.jar`, sind im Umfang von WebLoupe enthalten.

Analog zum TouchGraph implementiert die Klasse `de.webloupe.tree.UserObject` das Interface `GraphXMLNode`, das Methoden zur Gewinnung von Informationen für die Erstellung eines HyperGraph definiert.

Die öffentliche Methode, die zur Generierung einer GraphXML Datei und der Auswahl des Speicherorts dient, ist in der Klasse `HyperGraphExporter` enthalten. Sie verwendet die Utility Klasse `GraphXMLFactory`, die anhand des `TreeModels`, das ihrer öffentlichen Methode (die anderen sind privat) als Parameter übergeben wird, ein GraphXML Dokument erzeugt, das als String zurückgegeben wird.

Der `HyperGraphExporter` schreibt dann unter Verwendung der Utility Klasse `de.webloupe.utils.String2FileWriter` den String in eine Datei, deren Name und Ort vom Benutzer ausgewählt werden kann.

Zur Erstellung des GraphXML Dokuments greift die `GraphXMLFactory` auf Methoden der Klasse `GraphXMLBuildingBlocks` zurück, die Header, Footer, Knoten- und Kantenelemente in GraphXML Notation zurückgeben können.

Der Methode für Knoten werden eine ID und die URL (bei TouchGraph war es der Titel) übergeben, der Methode für Kanten werden die Quell- und Ziel-ID übergeben, der Header-Methode eine GraphID.

Zur Erstellung der Knoten und Kanten wird wie beim TouchGraph über das `TreeModel` iteriert, hier allerdings in der Klasse `GraphXMLFactory`, und die nötigen Attribute bei den `UserObjects` nachgefragt. Hier wird allerdings keine `HashMap` benötigt, da zur Erzeugung einer Kante die Quell- und die Ziel-ID ausreichen, und kein Elternknoten benötigt wird.

Weitere Informationen zu HyperGraph und Beispiele finden sie auf dessen Projektwebseite:

<http://hypergraph.sourceforge.net/>

### **3.3 Standard GUI**

Die Klassen für die Standard GUI sind bis auf die zum `JTree` gehörenden Klassen, die im Kapitel 3.2.1 erläutert werden, im Paket `de.webloupe.crawler.gui` untergebracht. Es hat folgende Inhalte:

### Klassen:

- `SettingsPanel`
- `StatisticsPanel`
- `StatusPanel`
- `TreeViewPanel`
- `WebLoupeControl`

Die Klasse `WebLoupeControl`, die auch die „main-Klasse“ der Applikation darstellt, ist von `javax.swing.JFrame` abgeleitet. Sie stellt die Kernklasse zur Anzeige von Komponenten und Benutzer-Steuerung des Crawlers dar. Alle anderen Klassen des Pakets sind von `JPanel` abgeleitet. Die `JPanels` werden alle in der Klasse `WebLoupeControl` erzeugt und entsprechend zusammengefügt.

Das `SettingsPanel` enthält `JTextFields` und Auswahlfelder wie `JSpinner` und `JComboBox`, um den Crawler zu konfigurieren und die Ansicht der Tabelle von Crawlergebnissen einzustellen. Sämtliche Einstellungen werden in einem `de.webloupe.crawler.Settings` Objekt verwaltet, die in Property-Dateien gespeichert werden. Siehe dazu im Kapitel 3.4.

Zusätzlich wurden Start- und Stop-Buttons für den schnellen Zugriff unterhalb der Einstellmöglichkeiten angebracht.

Das `StatisticsPanel` enthält die Tabelle mit den Crawlergebnissen. Weitere Informationen dazu im Kapitel 3.5.

Das `StatusPanel` wird horizontal am unteren Rand des `JFrames` angezeigt. Es enthält `JLabels`, die für das Feedback an den Benutzer benötigt werden. Dort werden Fortschritts-, Erfolgs- und Fehlermeldungen angezeigt, sowie die Anzahl gefundener, heruntergeladener, nicht erreichbarer URL's, und die Anzahl von Seiten, wo Keywords gefunden wurden. Sie werden während des Crawlens ständig aktualisiert. Dazu werden von der Klasse `de.webloupe.crawler.BreadthCrawlerMT` und seinen inneren Klassen Methoden der Klasse `StatusPanel` aufgerufen.

Des Weiteren enthält das `StatusPanel` eine `JProgressBar`, und ein `JLabel` zur Anzeige der Crawling-Zeit. Beides wird ebenfalls vom Crawler aus gesteuert. Für die Zeitanzeige wird ein `java.util.Timer` verwendet, der mit einem `java.util.TimerTask` ausgeführt wird. Dazu wurde die Klasse `CrawlingTimerTask` geschrieben, die eine innere Klasse des `StatusPanel` ist, und von `TimerTask` abgeleitet ist. Die `run()` Methode des `CrawlingTimerTask` wird jede Sekunde („Fixed Rate“) ausgeführt, um die Zeitanzeige des `JLabels` um eine Sekunde zu erhöhen.

Nach dem Ende des Crawlvorgangs werden die `JProgressBar` und die Zeitanzeige vom Crawler aus gestoppt. Wird der Crawler neu gestartet, werden alle Anzeigen des `StatusPanels` wieder auf Null gesetzt.

Im `TreeViewPanel` wird der `JTree` mit der Baumstruktur der Webseiten angezeigt. Weitere Informationen dazu im Kapitel 3.2.1.

Die Hauptkomponenten des `JFrames` `WebLoupeControl` werden in einem `BorderLayout`

angeordnet. Im Norden die `JToolBar`, im Zentrum in einem weiteren `JPanel` das Eingabefeld für die Start-URL und die `JTabbedPane`, und im Süden das `StatusPanel`.

In der `JTabbedPane` wird zu Beginn das `SettingsPanel` angezeigt. In weiteren Tabs sind einmal das `TreeViewPanel` und das `StatisticsPanel` in einer `JSplitPane` untergebracht, und in einem dritten Tab der `TouchGraph` in einem `GLPanel`, das eine `JPanel`-Klasse des `TouchGraph` Pakets `com.touchgraph.graphlayout` ist.

Beim Starten des Crawlers wird automatisch das Tab mit dem `JTree` ausgewählt, um den Crawlvorgang verfolgen zu können. Der `TouchGraph` und die Tabelle werden automatisch nach dem Crawlvorgang erzeugt und dargestellt.

Für die Aktionen des Menüs, der Toolbar und der Start- und Stop-Buttons im `SettingsPanel` wurden je Aktion Klassen von `javax.swing.AbstractAction` abgeleitet und als innere Klassen von `WebLoupeControl` implementiert.

- `StartAction` – zum Starten des Crawlers
- `StopAction` – zum Stoppen des Crawlers
- `ScreenshotAction` – zum Erzeugen eines Abzugs des aktuellen `TouchGraph` im PNG Format
- `HyperGraphExportAction` – zum Erzeugen einer GraphXML Datei der Baumstruktur

Alle Aktionen wurden mit Text, Icon, Tooltip, Accelerator und Mnemonic versehen. Sie werden im Konstruktor der jeweiligen Aktion definiert.

### **3.4 Properties und I/O**

Die Klassen für Properties und I/O wurden im Paket `de.webloupe.io` untergebracht. Es hat folgende Inhalte:

#### Klassen:

- `StringToFileWriter`

#### Unterpakete:

- `properties`

#### Klassen des Unterpakets:

- `PropertyManager`
- `SortedProperties`

#### Enums des Unterpakets:

- `AppPropertyKeys`
- `CrawlerPropertyKeys`

Die Klasse `StringToFileWriter` wird von der Klasse `de.webloupe.hypergraph.HyperGraphExporter` verwendet, um GraphXML Dokumente in

eine Datei zu schreiben. Weitere Informationen zu HyperGraph im Kapitel 3.2.4.

Die Klasse `PropertyManager` ist für das Lesen und Schreiben von Applikations- und Crawler-Properties zuständig. Dazu existieren zwei Property-Dateien im Verzeichnis „properties“ relativ zum Hauptverzeichnis von WebLoupe:

- `app.properties`
- `crawler.properties`

In den Applikations-Properties werden Einstellungen gespeichert, die die Darstellung der Anwendung betreffen:

- Größe der Applikations-Fensters
- Position des Applikations-Fensters
- Anzeige von bestimmten Spalten in der Tabellenansicht (Benutzereinstellungen im `SettingsPanel`; `JComboBoxes`)
  - URL's
  - Titel
  - HTTP Response Code
  - Status
  - Mime Typs
  - Anzahl der gefundenen Zeichen
  - Anzahl der gefundenen Bilder
  - Dateigröße
  - gefundene Keywords

In den Crawler-Properties werden vom Anwender definierbare Parameter des Crawlers gespeichert:

- max. Anzahl der Webseiten, die gesammelt werden sollen
- max. Tiefe der Ebenen
- Einschränkung des Suchbereichs (Enum `de.webloupe.crawler.Scope`):
- Einschränkung des Mime Typs (Enum `de.webloupe.crawler.MimeType`)
- max. Zeit, nach der ein Verbindungsversuch zu einem Server abgebrochen wird
- max. Dateigröße von Ressourcen

Details zu diesen Parametern wurden bereits im Kapitel 3.1 geschildert.

Das Speichern und Lesen von diesen Properties wird durch Methoden der Klasse `java.util.Properties` unterstützt. Jedoch wird diese Klasse nicht direkt verwendet, sondern eine Subklasse `SortedProperties` des Pakets `de.webloupe.io.properties`, durch die Properties in alphabetischer Reihenfolge in die Dateien geschrieben werden. Bei einer größeren Anzahl von Properties kann das die Überprüfung bei der Programmierung vereinfachen.

Beim Start der Applikation werden die Properties aus den Dateien eingelesen und den Attributen eines `de.webloupe.crawler.Settings` Objekts zugewiesen. Fehlen bestimmte oder alle Properties oder gar die Dateien selbst, werden Standardwerte verwendet, die in der Settingsklasse

definiert sind.

Nachdem die Werte eingelesen wurden, wird die GUI erzeugt, die die Werte des Settingsobjekts als Standardwerte übernimmt.

Wird die Applikation beendet, werden die aktuell eingestellten Werte wieder als Properties in die Dateien geschrieben. Sind die Dateien aus irgendwelchen Gründen nicht vorhanden, werden sie vorher erzeugt.

Die Schlüssel der Properties, die zum Lesen und Speichern benötigt werden, sind in den Enums `AppPropertyKeys` und `CrawlerPropertyKeys` festgelegt.

### 3.5Utils

Die allgemeinen Utility-Klassen wurden im Paket `de.webloupe.utils` untergebracht. Es hat folgende Inhalte:

#### Klassen:

- `BrowserLauncher`
- `Debug`
- `UrlUtil`

Die Klasse `BrowserLauncher` ist freie Software zum Öffnen von URL's im Standard-Webbrowser eines Betriebssystems, und kann unter <http://browserlauncher.sourceforge.net/> (zuletzt gesehen am 10.2.2005) heruntergeladen werden. Sie wird im Zusammenhang mit dem `JTree` eingesetzt, um die URL eines Knotens nach Doppelklick öffnen zu können. Dazu muss einfach die statische, öffentliche Methode `BrowserLauncher.openURL(String)` aufgerufen werden, mit der URL als Parameter. Weitere Informationen zum `JTree` im Kapitel 3.2.1.

Die Klasse `Debug` hat zwei statische Methoden zur Ausgabe von beliebigen Strings, die als Parameter übergeben werden. Mit der Methode `printMessage(String)` kann eine Nachricht auf der Konsole ausgegeben werden. Die Methode `writeLogFile(String, String, boolean)` schreibt eine Nachricht in eine Datei. Es gibt noch eine dritte Methode, `deleteLogFile(String)`, mit der man eine Datei wieder löschen kann.

Der Vorteil bei der Nutzung der `Debug` Klasse besteht darin, dass sie ein öffentliches bool'sches Flag besitzt, mit dem man die Ausführung der Methodenrumpfe verhindern kann. D.h. alle Ausgaben können im Code bestehen bleiben, man braucht nur das Flag auf `false` zu setzen, und die Nachrichten werden nicht mehr ausgegeben.

Die Klasse `UrlUtil` des Pakets `de.webloupe.utils` wird für folgende Zwecke eingesetzt:

- Überprüfen und Vervollständigen der Starturl – es wird nur das `file` und das `http` Protokoll unterstützt; Vervollständigen heißt z.B. „`http://`“ voranzustellen, wenn die URL mit `www`.

beginnt

- Überprüfung des `scope` einer URL
- Überprüfen, ob eine URL unterstützt wird – Javascript und Mailto URS's werden verworfen
- Erzeugen einer absoluten URL aus einer eingebetteten (und evt. relativen) URL und der Base URL der Webseite, in der die eingebettete URL gefunden wurde

## 4Ausblick

Der Programmcode von WebLoupe wurde Mitte Februar in der Version 0.5 auf Sourceforge.net veröffentlicht. Er enthält alle Funktionen, die in dieser Entwicklerdokumentation beschrieben sind. Für eine detaillierte Beschreibung der Implementation empfiehlt sich ein Blick in die API Dokumentation von WebLoupe, und natürlich in den Programmcode.

Es wurde versucht, Fehler im Programmcode weitgehend zu vermeiden und sämtliche Funktionen getestet. Dennoch können Programmfehler nicht ausgeschlossen werden. Bitte nutzen sie die Möglichkeit, und veröffentlichen etwaige Fehler (Bugs) auf der Sourceforge.net Projektwebseite von Webloupe: <http://sourceforge.net/projects/webloupe/>

Es sind auch Vorschläge für Erweiterungen (sog. Feature Requests) willkommen, oder Fragen zur Benutzung oder Entwicklung von WebLoupe (sog. Support Requests). Bitte nutzen sie die Möglichkeiten von Sourceforge.net auf oben genannter Projektwebseite.

Wollen sie an der Entwicklung von WebLoupe mitwirken, kontaktieren sie uns bitte unter einer der folgenden Emailadressen:

[webspirit@users.sourceforge.net](mailto:webspirit@users.sourceforge.net) (Programmierung)

[kaimka@users.sourceforge.net](mailto:kaimka@users.sourceforge.net) (Marketing, Design)

Wünschenswert wären unter anderem folgende Erweiterungen von WebLoupe:

- Öffnen des Standardbrowsers nach Doppelklick auf einen TouchGraph Knoten
- direkte Integration von HyperGraph zur Visualisierung als Alternative zum TouchGraph
- Erstellen einer Report-Datei über die Ergebnisse des Crawlvorgangs
- Speichern und Öffnen von Projektdateien, mit denen sich die Baumstruktur abspeichern lässt, um nach dem Öffnen wieder den JTree, die Tabelle und den TouchGraph anzeigen zu können
- Interpretation von robots.txt

Ausserdem wurden beim Crawlen einer größeren Anzahl von Webseiten, z.B. ab 500, schon gewisse Instabilitäten beobachtet. Dies muss weiter beobachtet, und falls nötig und möglich, behoben werden.

## 5Quellenangaben

[1] SUN Microsystems - Java 5.0 in a Nutshell:

<http://java.sun.com/developer/technicalArticles/releases/j2se15/#misc>,

zuletzt gesehen am 15.10.2004