Entwickler - Dokumentation

Michael Bindek
Timo Ladberg
Andreas Valentin

Fachhochschule Kaiserslautern, Wintersemester 2006/2007, Prof. Hendrik Speck
http://moleicious.sourceforge.net

Inhaltsverzeichnis

1 Einleitung....3
1.1 Mole.icio.us....3
1.2 Lizenz und Disclaimer....3
1.3 Weiterentwicklung....3
1.4 Entwicklerdokumentation....4
1.5 Installationsanleitung unter Eclipse....4
2 Entwicklungsumgebung....5
2.1 Systemvoraussetzungen....5
2.2 Programmiersprachen....5
2.3 Entwicklungsumgebung....5
3 Softwarekomponenten....6
3.1 Data-Harvester....6
3.2 Package stuff....7
3.3 Package moler....7
3.4 Package Parser....8
3.5 Package moleThreads....9
3.6 Package threadExceptions....10
3.7 Package moleDB....10
3.8 Package moleDTO....10
3.9 Package fixDBProblems....11
3.10 Visualisierung....11
3.11 Generierung der Graphen....13
3.12 Generierung der VisualizationViewer....13

3.13 Lokales Speichern/Laden eines Users....14
3.14 Package MoleVizu....14
3.15 Package moleVertex und moleEdge....14
3.16 Package moleLayout....15
3.17 package moleDecorator....15
3.18 Package graphMaker....15
4 Klassendiagramme....16
5 Datenbank....21

1 Einleitung

1.1 Mole.icio.us

Mole.icio.us ist eine Open Source Software zum Analysieren und Visualisieren der Online Bookmark Sammlung „del.icio.us“. Die hierfür benötigten Daten werden zunächst mit einer entsprechenden Applikation (Crawler) gesammelt. Anschließend werden diese Daten gesplittet (Parser) und so aufbereitet in eine Datenbank geschrieben. Das Visualisierungs-Tool verarbeitet die Daten und stellt sie in verschiedenen Graphen dar. Es werden die Beziehungen zwischen den einzelnen Nutzern und ihrem Netzwerk sowie ihre gespeicherten Tags (Beschreibungen der Bookmarks) und deren Beziehungen
untereinander dargestellt. Über ein Informationsfeld werden zusätzliche Daten über diese Beziehungen angezeigt. Zudem ist es möglich, durch das Ein- und Ausschalten von verschiedenen Filtern, die Graphen zu modifizieren.
Das Mole.icio.us Projekt wurde im Rahmen der Lehrveranstaltung Medienkonzeption- und Produktion im Wintersemester 2006/2007 an der Fachhochschule Kaiserslautern, Standort Zweibrücken, gestartet. Aufgabe war es, eine Software zur Visualisierung eines sozialen Netzwerkes zu entwickeln. Die Hauptmerkmale wurden hierbei auf die folgenden drei Punkte gelegt:

- Sammeln der Informationen mittels eines Crawlers
- Speichern der Daten in einer Datenbank
- Auswertung und Anzeigen der geforderten Informationen mittels eines entsprechenden Algorithmus

Um eine Weiterentwicklung der Software zu gewährleisten, sollte sie unter einer Open Source Lizenz realisiert und auf der SourceForge.net Seite zugänglich gemacht werden.

1.2 Lizenz und Disclaimer

Mole.icio.us wird unter der GNU General Public License (GPL) entwickelt. Mehr Informationen hierzu finden Sie unter http://www.opensource.org.
Mole.icio.us übernimmt keine Haftung für eventuelle Schäden, die durch Nutzung bzw. Weiterentwicklung des Crawlers oder der Visualisieung entstehen könnten.

1.3 Weiterentwicklung

Durch eine Zusammenarbeit mit „del.icio.us“ bzw. „Yahoo!“ wäre eine einfachere Auswertung der Daten aus deren Datenbank möglich. Dadurch würde die Crawler-Komponente erheblich entlastet und man könne sich bei der Visualisierung auch Graphen mit mehreren Ebenen zuwenden.

1.4 Entwicklerdokumentation

Ziel dieser Dokumentation ist es, Entwicklern bei der Weiterentwicklung bzw. Verwendung einzelner Teile des Quellcodes zu unterstützen. Dieses Dokument dient zum besseren Verständnis der Gesamtstruktur der Anwendung. Grundvoraussetzung zum verstehen dieser Dokumentation sind Grundkenntnisse in Java 5.0. Darüber hinaus wird empfohlen, sich in das Java Universal Network/Graph Framework (JUNG) einzuarbeiten.

1.5 Installationsanleitung unter Eclipse

Nachdem das Projekt über CVS herunter geladen und in den Eclipse – Workspace eingebunden wurde, müssen zusätzliche Libraries bekannt gegeben werden. Diese befinden sich im lib – Ordner des Projektes. Beim Anlegen des Projektes ist darauf zu achten dieses als Java – Projekt zu deklarieren.

Zum Einbinden einer eigenen Datenbank müssen in der db.properties – Datei folgende Werte eingegeben werden:

```
##database-url
url=jdbc:mysql://"Ihr Host"/moleicious
##username
user="Ihr Username"
##password
password="Ihr Passwort"
```

Es wird dringend empfohlen, die Datei „moleicious_db.zip“ in Ihre Datenbank zu importieren. Diese enthält ein Skript zum Anlegen der Datenbank sowie die Grundstruktur der benötigten Tabellen.

Zur initialen Bestückung der Datenbank empfehlen wird das Einlesen der mitgelieferten RSS – Datei. Nähere Informationen hierzu erhalten Sie unter Kapitel 3.3.

2 Entwicklungsumgebung

2.1 Systemvoraussetzungen

Zur Weiterentwicklung bzw. Nutzung von Teilen des Quellcodes von Mole.icio.us sind folgende Komponenten notwendig:

- Java Development Kit JDK 5.0 der Java Standard Edition J2SE
- MySQL Datenbank

2.2 Programmiersprachen

Die gesamte Anwendung wurde in Java 5.0 programmiert, wobei für die Visualiesierungskomponente das Jung Framework verwendet wurde.

2.3 Entwicklungsumgebung

Das Projekt wurde mit Eclipse und dem CVS Plugin entwickelt, wodurch ein ständiger Versionsabgleich möglich war. Die SQL Abfragen wurden mit phpMyAdmin erstellt und dann in Eclipse in die entsprechenden Java Klassen übernommen.

3 Softwarekomponenten

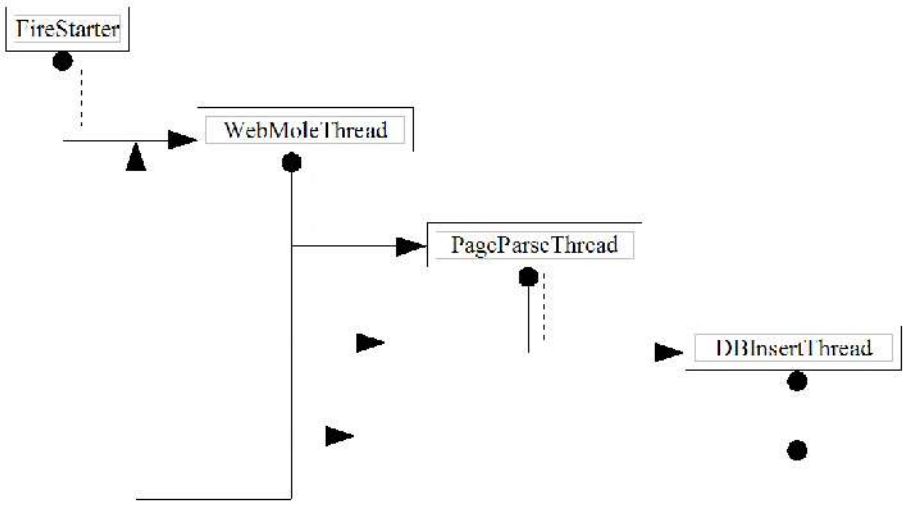
3.1 Data-Harvester

Der Harvester besitzt drei grundlegende Funktionen:

Das systematische Crawlen von HTML-Seiten, die Extraktion relevanter Daten mit einem Parser und das Speichern dieser Informationen in einer relationalen Datenbank.
Crawler-, Parser und Datenbankzugriffskomponenten sind in separate Threads implementiert und werden verschachtelt in der Reihenfolge Crawler -> Parser -> Datenbankzugriff aufgerufen.

Grundlegende Funktionsweise:

Initial wird ein Objekt der Klasse DataDonkey erstellt. Die Klasse holt sich Informationen über noch ungecrawlte Nutzer aus der Datenbank und beinhaltet weitere Methoden, die die Übergabe von Daten zwischen Crawler, Parser und Datenbankabfragen steuern.
Dieses Objekt wird an einen oder mehrere Crawler-Threads übergeben, die jeweils einen User crawlen. Es werden sukzessiv HTML Seiten und JSON-Feeds eines einzelnen Users heruntergeladen, der mit der Methode *getNextUnmoledUser()* aus dem DataDonkey ermittelt wird. Nach jedem erfolgreichen HTML Zugriff wird ein Parse-Thread angestoßen, der die notwendigen Daten aus dem Dokument herausfiltert. Nach erfolgreichem auslesen stößt dieser Parse-Thread wiederum einen Thread an, der die gewonnenen Informationen in die Datenbank schreibt.
Für die JSON-Feeds werden die Parser-Threads im Crawler-Thread selbst (WebMoleThread), für die einzelnen HTML Seiten in der Methode *GetSingleUserPage()* aufgerufen.
Der Crawler-Thread arbeitet währenddessen weiter und lädt sukzessiv weitere HTML Dokumente herunter und verfährt nach erfolgreichem Download im selben Schema. Sobald ein User vollständig gecrawlt wurde, wird rekursiv ein neuer Crawler-Thread gestartet und das DataDonkey Objekt übergeben.



Verwenden von Proxy Servern

Während den ersten Crawldurchgängen ergab sich das Problem, dass durch die permanenten Anfragen, die von dem crawlenden Client ausgingen, schnell eine Überlastung der Netzwerkressourcen auftrat. Das Anfordern der Pages überstieg die uns zur Verfügung stehenden Traffic-Kapazitäten und wir mussten uns nach Alternativen umsehen, mit denen wir dieses Problem umgehen konnten.
Als einzig sinnvolle Lösung erschien das Verwenden von Proxyservern und das sich daraus ergebende verteilte Crawlen von mehreren Maschinen. Dies hat den Vorteil, dass es nicht zu einer Überlastung des Netzwerkes kommt und dass die benötigten Seiten innerhalb kürzester Zeit abgerufen werden können.
Die Proxyserver werden in der Klasse Crawler.java angegeben und können hier je nach Auslastung und Verfügbarkeit ausgetauscht werden.

3.2 Package stuff

DataDonkey.java

Diese Klasse nimmt Verbindung zur Datenbank auf und speichert eine Liste ungecrawlter User. Ferner werden hier Methoden bereitgestellt, die die Übergabe der Daten von Crawler zu Parser und von Parser zu Datenbankklassen ermöglichen.
Es wird ein DataDonkey Object erstellt und dann jedem WebMoleThread als Parameter übergeben.

FireStarter.java

Hier wird ein DataDonkey Objekt erstellt und die WebMoleThreads werden gestartet.

3.3 Package moler

Crawler.java

Hier finden sich die Methoden, die HTTP - Verbindungen erstellen und HTML Daten bzw. JSON Feeds herunterladen. Beim jeweiligen Download bestimmt der Identifikator des Threads den Proxy, der zwischengeschaltet wird.
Um klar aufzuzeigen, dass es sich nicht um einen gewöhnlichen manuellen HTTP - Zugriff handelt wird ebenfalls als UserAgent „moler 0.3 -- www.fh-kl.de --“ übergeben.
getUserFans und getUserNetwork sind für die JSON-Feeds zuständig und übergeben im Falle eines missglückten Downloads einen NULL - Wert.
getSingleUserPage wird von den Methoden *getAllUsersPages* und *getTheRestOfTheUsersPages* verwendet und lädt eine einzelne HTML - Seite herunter. Im Unterschied zu den JSON - Feed bezogenen Methoden wird hier bei Erfolg direkt ein Parse - Thread über die im dataDonkey - Objekt verfügbaren Methoden gestartet.
Die zuletzt erfolgreich gecrawlte Seite wird hier im UserPageDTO - Objekt vermerkt, um

bei einem möglicherweise folgenden Fehler an dieser Position ansetzen zu können.

Es wird die Anzahl der HTML Seiten des entsprechenden Users ermittelt und diese dann durch das Aufrufen von *getSingleUserPage* heruntergeladen.
getAllUsersPages wird in WebMoleThreads aufgerufen, *gettheRestOfTheUsersPages* wird zum Abschließen eines missglückten Crawlvorgangs in *screwedUpWebMoleThreads* aufgerufen.

GetRSS.java

Zum initialen Bestücken der Datenbank ist es notwendig, eine RSS-Datei einzulesen. Diese befindet sich im Package „moler“. Mit der Klasse GetRSS kann diese Datei ausgewählt und anschließend in die Datenbank geschrieben werden. Das Verfahren arbeitet ähnlich dem in der Klasse „Parser“. Die Usernamen werden durch Aufspalten der RSS - Datei ausgelesen und an eine Collection übergeben. Diese wird anschließend durchlaufen und deren Einträge werden in die Datenbank geschrieben. Erst jetzt ist gewährleistet, dass beim Aufruf der Klasse „Crawler“ keine falschen Informationen zum Abrufen der einzelnen Userseiten eingelesen werden.

3.4 Package Parser

Parser.java

Die grundlegende Funktion des Parsers besteht darin, die gesammelten Informationen in kleinere Teilstücke aufzuspalten und diese zur weiteren Verarbeitung an die entsprechenden Klassen zu übergeben.
Den ersten Schritt übernimmt dabei die Methode *getSubstring()*. Diese erhält als Übergabeparamenter eine Zeichenkette – die eigentliche Seite eines Users. Da nicht alle Informationen dieser Zeichenkette relevant sind, wird diese zunächst grob aufgespalten und jedes einzelne Teilstück an eine Collection übergeben, auf die im Folgenden genauer eingegangen wird.
Nachdem nun alle Substrings in der Collection gespeichert wurden, wird diese anschließend durch iteriert und es werden für jedes Element die daraus benötigten Informationen entnommen. Dies geschieht mittels der Methoden *getSubstringBookmark()*, *getSubstringTags()*, *getSubstringDate()*, *getSubstringHash()* und *getStubstringNumberOfUser()*.
Neben den Daten, die aus den Substrings gewonnen werden, benötigt die Klasse noch weitere Informationen zum Netzwerk jedes einzelnen Users, der Nummer der aktuell gecrawlten Seite und der Anzahl der noch verbleibenden Seiten. Diese werden mittels der Klassen *getUserNetwork()*, *getNumberOfPagesToCrawl()* und *getNumberOfActualPage()* gewonnen. Hier wird nicht ein Substring, sondern die aus den JSON Feeds ausgelesene Seite übergeben.

Nun liegen alle benötigten Daten vor. In der Methode *getAllSubstringInfo()* werden diese anschließend zusammengetragen und an dafür vorgesehene DTO-Objekte übergeben.

Abschließend werden alle DTO-Objekte in der Methode *parseThePage()* an eine Collection übergeben, die nun sämtliche Informationen über einen User, die Anzahl seiner Seiten und die darauf befindlichen Bookmarks inkl. Tags und Zusatzinformationen enthält.

3.5 Package moleThreads

WebMoleThread.java

Wählt mit der Methode *datadonkey.getNextUnmoledUser()* einen User aus und lädt nacheinander erst JSON - Feeds, die die Netzwerke dieses Users beinhalten, und dann die entsprechenden HTML Seiten herunter.

Nach jedem erfolgreichen HTTP - Zugriff wird der entsprechende Parser als Thread gestartet. Für die JSON - Feeds geschieht dieser Aufruf explizit im WebMoleThread, für HTML Seiten in *GetSingleUserPage()*. Während die benötigten Informationen aus einem Dokument ausgelesen werden, läuft der Crawler parallel weiter.
Jeder WebMoleThread erhält als Parameter einen Identifikator der sicherstellt, dass verschiedene Proxys pro Instanz benutzt werden (siehe Kapitel Proxies).
Sobald ein User vollständig gecrawlt wurde und keine Fehler aufgetreten sind, wird rekursiv ein neuer WebMoleThread aufgerufen, der wiederum einen neuen User crawlt.

Falls ein HTTP - Zugriff nicht erfolgreich war, wird die aktuelle Postion vermerkt und ein entsprechender BOOLEAN - Wert im UserPageDTO gesetzt. Eine Abfrage dieser Werte hat in diesem Fall eine Exception zur Folge. Es wird ein ScrewedUpWebMoleThread gestartet, der den Vorgang zu Ende führt.
Unhandled Exceptions werden in dem Package threadExceptions behandelt.

ScrewedUpWebMoleThread.java

Die Klasse führt einen missglückten Crawl eines Users anhand von DTO's, die im Konstruktor übergeben wurden nach dem selben Schema zu Ende.

PageParseThread.java

Hier wird der Parser aufgerufen, der die heruntergeladenen Dokumente aufspaltet und die gewünschten Informationen heraus liest. Anschließend wird mit der Methode *insertIntoDB()* des DataDonkey - Objekts ein Thread zum Einspeisen der Daten in die Datenbank gestartet.

NetworkParseThread.java & FansParseThread.java

Analog zu PageParseThread.java werden die heruntergeladenen JSON - Feeds analysiert und in die Datenbank geschrieben.

DBInsertThread.java

Diese Klasse wird immer dann aufgerufen, wenn entweder das Interessen - Netzwerk, das Fan - Netzwerk oder eine Seite eines Users geparst wurde. Hier wird dann jeweils eine Verbindung zur Datenbank hergestellt und die entsprechenden Funktionen aus dem moleDB - Package zum Schreiben der Daten in die Datenbank werden aufgerufen.

3.6 Package threadExceptions

Die Klassen in diesem Package fangen unhandled Exceptions des WebMoleThread und ScrewedUpWebMoleThread ab und starten diese neu.

3.7 Package moleDB

DBObjekt.java

In dieser Klasse wird durch die Funktion *getConnection()* eine Verbindung zur Datenbank hergestellt.

UserDB.java, BookmarkDB.java und TagDB.java

Durch die hier enthaltenen Funktionen können mittels SQL-Abfragen neue User, Bookmarks bzw. Tags in der Datenbank angelegt und deren Informationen wieder abgerufen werden. In UserDB.java sind zusätzlich die Funktion zum Abspeichern der „Interests“ bzw. „Fans“ Beziehungen und der Beziehungen der User zu den Bookmarks und Tags enthalten.

3.8 Package moleDTO

UserDTO.java, BookmarkDTO.java, TagDTO.java und UserBooksDTO.java

Um die Daten an die DB Klassen zu übergeben bzw. die aus den DB Klassen gewonnenen Daten abzuspeichern sind hier Methoden zum Erstellen von Data Transfer Objekten enthalten. Entsprechend der Namen werden in einem UserDTO Daten der User, in einem

BookmarkDTO Daten der Bookmarks, in einem TagDTO Daten der Tags und in einem UserBooksDTO Daten der Beziehungen zwischen User, Bookmarks und Tags gespeichert.

UserPageDTO.java

Die zum Crawlen benötigten Informationen über den aktuellen Fortschritt werden in einem UserPageDTO gespeichert. Hier wird hinterlegt, wie viele Seiten eines Users schon gecrawlt wurden und wie viele es insgesamt sind. Außerdem wird gespeichert, welche Schritte des Crawlvorgangs schon abgewickelt wurden und ob der Crawlvorgang evtl. unterbrochen wurde.

3.9 Package fixDBProblems

UpdateDB.java

Um die Datenbank mit zusätzlichen Informationen zu den einzelnen Usern zu bestücken, werden hier die Anzahl der Bookmarks, Tags und Verbindungen zu anderen Useren ermittelt und in die Datenbank geschrieben. Die Anwendung greift dabei auf Klassen im moleDB - Package zu, die die jeweiligen Werte aus der Datenbank ermitteln und zurückgeben.

3.10 Visualisierung

Kurzübersicht

In dieser Klasse MoleVisu.java wird das komplette Interface mit Hilfe von AWT und SWING erstellt. Hier werden ebenfalls die in der CreateGraphs.java generierten JUNG-Graphen modifiziert und eingebunden.

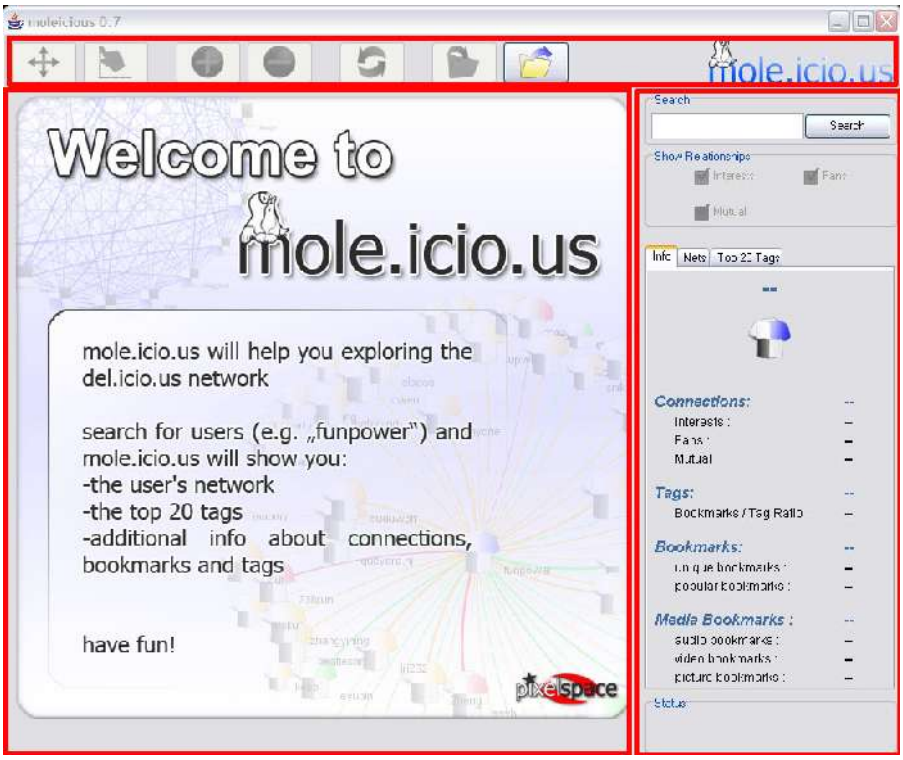
Oberfläche

Der Skin der Oberfläche passt sich durch den Aufruf

„UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());“

automatisch an die Systemoberfläche an.

Primär ist die Oberfläche in drei Teile gegliedert, die im BorderLayout angeordnet sind:



Norden:
JPanel topline entspricht der Button-leiste die über die Methode *getControls()* eingebettet wird und dem Mole.icio.us - Logo.

Westen:
Welcome Panel (*getSplashPanel()*), der nach der ersten Suche durch eine TabbedPane, in denen die beiden Graphen eingebettet sind (networkAndTagsTabbedPane), ersetzt wird. Der Aufruf wird im actionListener „al" bereitgestellt, der mit dem Bestätigen der Suche gestartet wird.
networkAndTagsTabbedPane wurde ebenfalls mit einem changeListener versehen, der je nach Tab den graphoptions - Panel neu besetzt.

Osten:
JPanel im BorderLayout, das drei Komponenten beinhaltet :

Norden:	JPanel mit Suchmaske(*getSearch()*) und Graphenkontrolle (*getGraphControls()* -> JPanel graphoptions)
Zentrum:	Verschachteltes JPanel (*getInfo()*) mit einer JTabbedPane in dem Statistiken und JTree Repräsentationen der Graphen zu finden sind.
Süden:	Statusanzeige (*getProgressBar()*);

3.11 Generierung der Graphen

Nachdem eine Suchabfrage abgeschickt wurde, werden in der Klasse CreateGraphs.java zwei Instanzen von UndirectedGraph erstellt. Aus der Datenbank werden anschließend entsprechende Informationen über Netzwerke, Bookmarks und Tags ausgelesen.
Diese werden in die Graphen als Nodes, Edges und UserData angelegt.

Nach der Fertigstellung werden diese Graphen in der Klasse MoleVisu.java über die Funktionen *MoleVisu.setGraph(g) und MoleVisu.setGraph2(g2)* gesetzt.

Anschließend werden durch die Funktionen *drawGraph()* und *drawGraph2()* entsprechende Layouts erstellt.
Nodes und Edges der Graphen werden ausgelesen und in Sets gespeichert, um später ein Ein- und Ausblenden verschiedener Nodes und Edges der Graphen zu ermöglichen.

Die Methoden *CenterMainNode(), AdjustNodes() und setVVScale()* ordnen die Nodes und Edges des Networks Graphen an.
Die Methode *CenterMainNode()* zentriert den Hauptuser im Graphen, *AdjustNodes()* setzt den Abstand fest, der zwischen dem Hauptuser und dem jeweiligen User seines Netzwerks eingehalten werden soll. *setVVScale()* skaliert den Graphen entsprechend seiner Größe, um ihn ganz im Viewer darstellen zu können.

3.12 Generierung der VisualizationViewer

Die Erstellung der Viewer wird in den Methoden *getLeftGraphPanel()/getRightGraphPanel()* und *getLeftViewer()/getRightViewer()* durchgeführt.

Ein VisualizationViewer wird für jeden Graphen erstellt und mit einem Layout des entsprechenden Graphen (erstellt in *drawGraph()* bzw. *drawGraph2()*) und einem Renderer initialisiert.
In weiteren Schritten werden MouseListener, ToolTipFunctions und weitere Funktionen die das Erscheinungsbild und gewünschte Verhalten an den VisualizationViewer übergeben.

Genauere Spezifikationen und detaillierte Beschreibungen zur Erstellung können der JUNG Dokumentation unter

http://jung.sourceforge.net/doc/index.html

entnommen werden.

3.13 Lokales Speichern/Laden eines Users

Ein User kann lokal in Form von GraphML Dateien im XML - Format gespeichert und geladen werden.
Hierbei wurde auf den JFileChooser als Interface zurückgegriffen.

Die entsprechenden Funktionen sind in den actionListenern der Buttons graphSave respektive graphLoad implementiert.

3.14 Package MoleVizu

MoleVisu.java

Dies ist die zentrale Klasse, die die Oberfläche mit den meisten Funktionen der Visualisierung beinhaltet.

Grundlegende Aspekte der Funktionsweise werden in den Abschnitten Oberfläche, Generierung des Graphen und Generierung der VisualizationViewer erläutert.

Einige Funktionen, wie der Aufruf von *createUserGraph (actionListener „al")* und das Zentrieren von Nodes nach *DoubleClick (TestGraphMouseListener)* wurden als Threads implementiert um ein paralleles Weiterarbeiten zu ermöglichen.
Da Swing und AWT nicht als threadsafe gelten, kann es in diesem Zusammenhang vereinzelt zu Exceptions kommen, die aber keine auffälligen Auswirkungen auf den Rest der Applikation haben.

MoleVertexIcon.java

Diese Klasse implementiert VertexIconFunction und legt fest, welche Icons in den Graphen zur Darstellung der Nodes verwendet werden sollen.

MoleToolTip.java

Hier wird der Inhalt von Tooltips, Nodes und Edges geregelt.

3.15 Package moleVertex und moleEdge

Diese Packages beinhalten Klassen, die von UndirectedSparseVertex und UndirectedSparseEdge erben bzw. erweitert werden.

3.16 Package moleLayout

moleLayout.java

Erweitert CircleLayout. Die Funktion orderVertices wurde einfach überschrieben um die zufällige Neuanordnung von Elementen der Graphen bei der Reset - Funktion zu unterbinden.

3.17 package moleDecorator

moleEdgePaint.java

Hier wird die Farbe der Edges anhand ihres UserDatums „weight“ festgelegt.

moleEdgeStroke.java

Hier wird die Dicke der Edges anhand ihres UserDatums „sharedbooksweight“ festgelegt.

moleVertexStringer.java

Diese Klasse sorgt dafür, dass Labels von Nodes nach 15 Stellen mit „...“ abgekürzt werden.

3.18 Package graphMaker

CreateGraphs.java

Diese Klasse generiert Graphen anhand der Netzwerke und Daten, die von der Datenbank bereitgestellt werden.

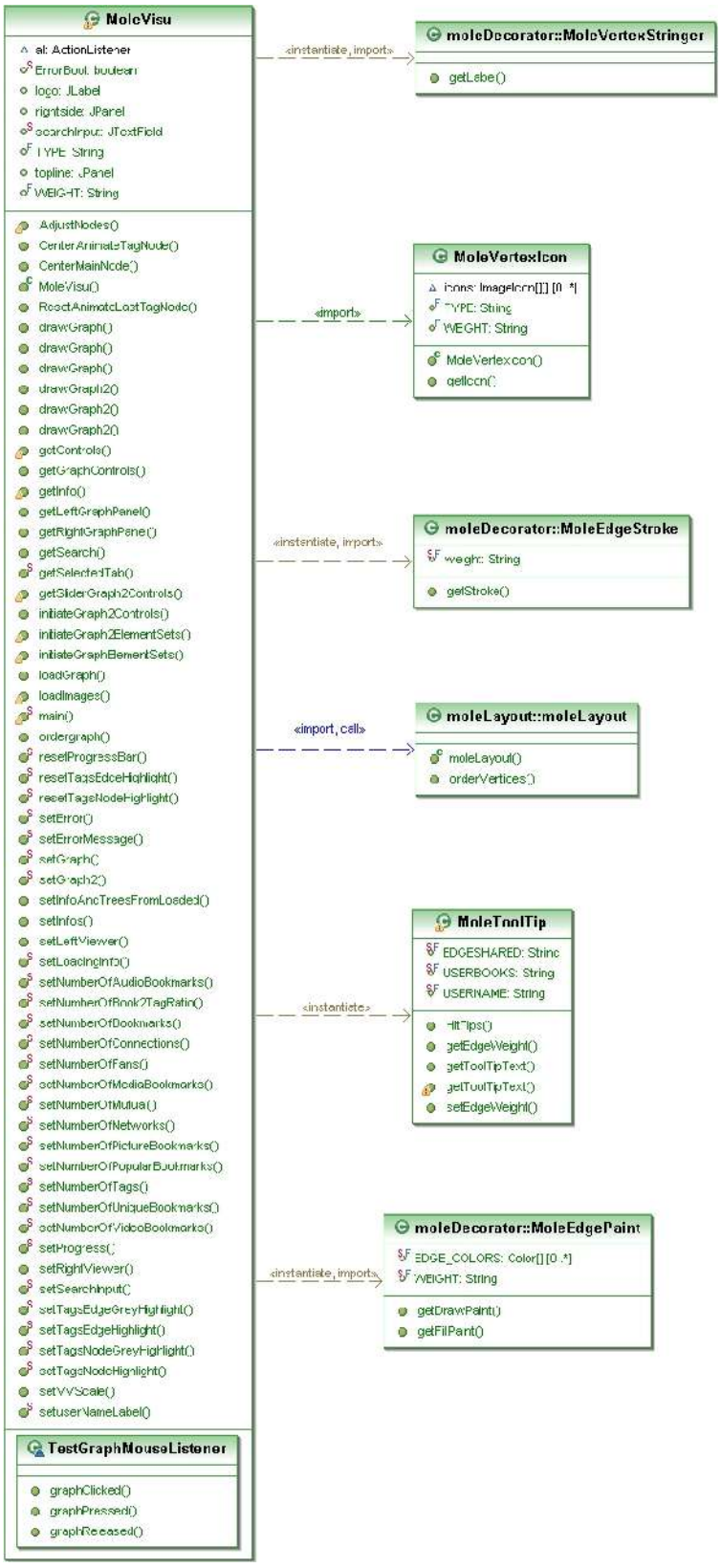
Genauere Spezifikationen und detailierte Beschreibungen zur Erstellung eines Graphen mithilfe des JUNG Toolkits können unter

http://jung.sourceforge.net/doc/index.html

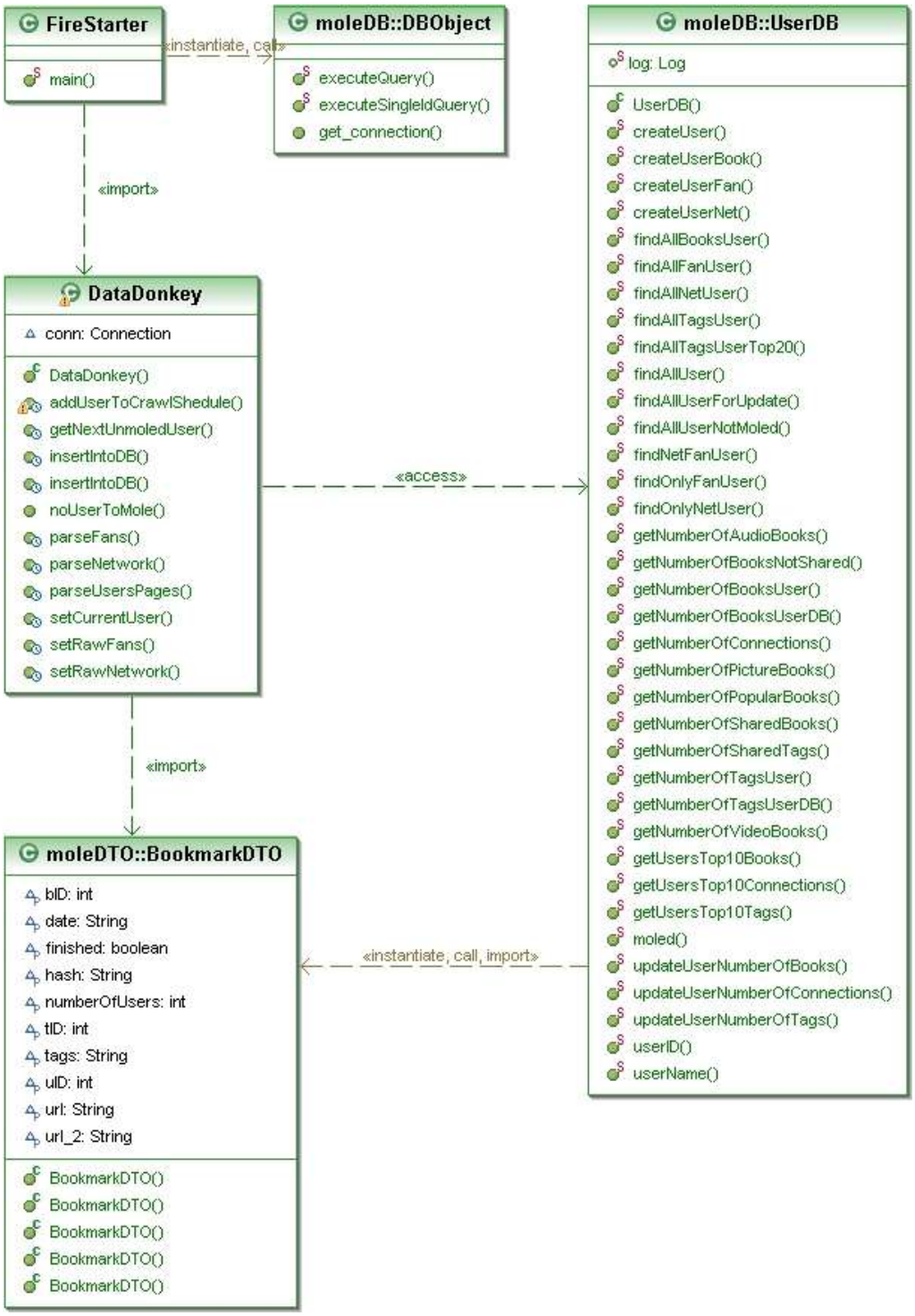
entnommen werden.

4 Klassendiagramme

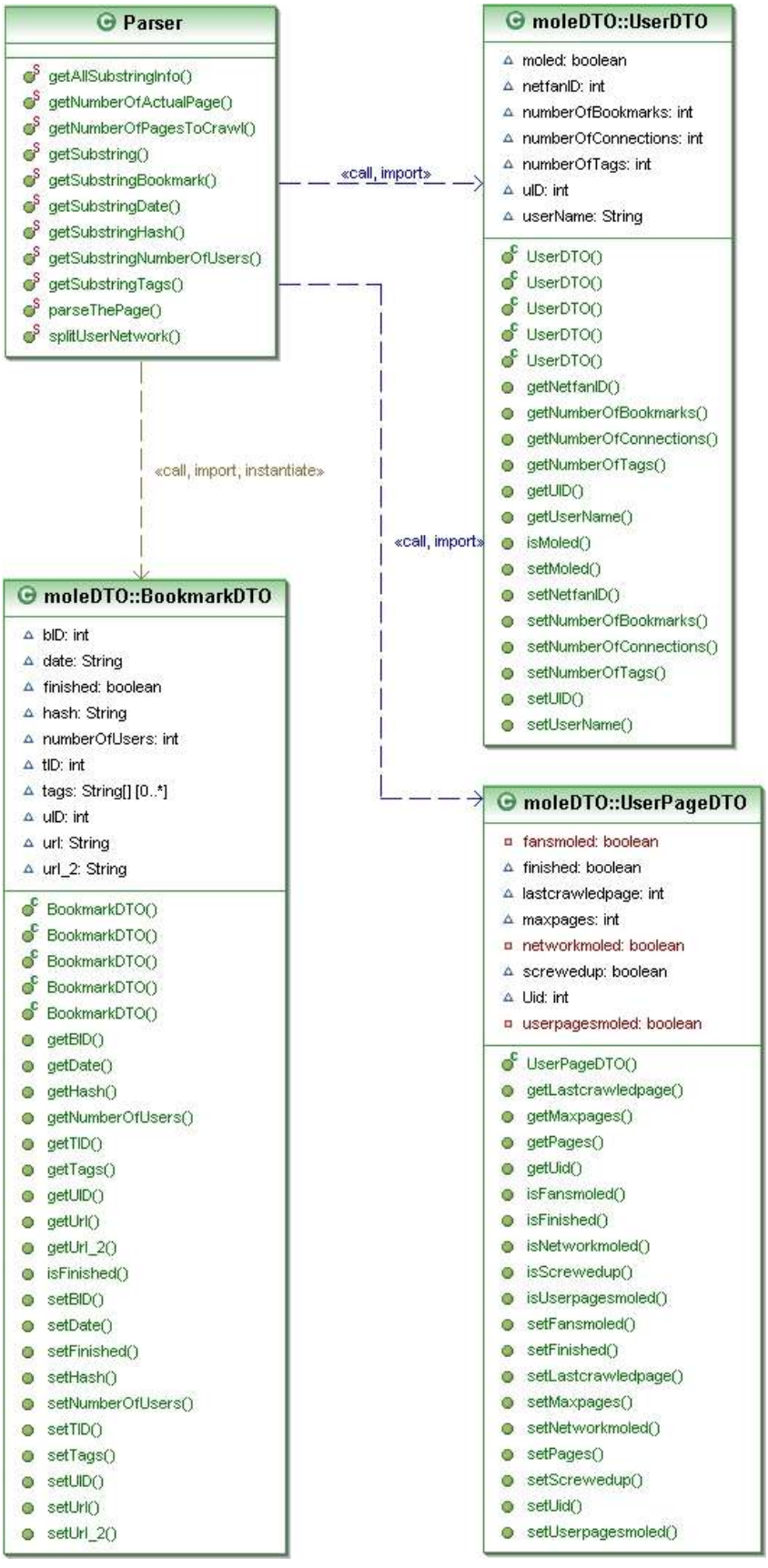
Klassendiagramm 1



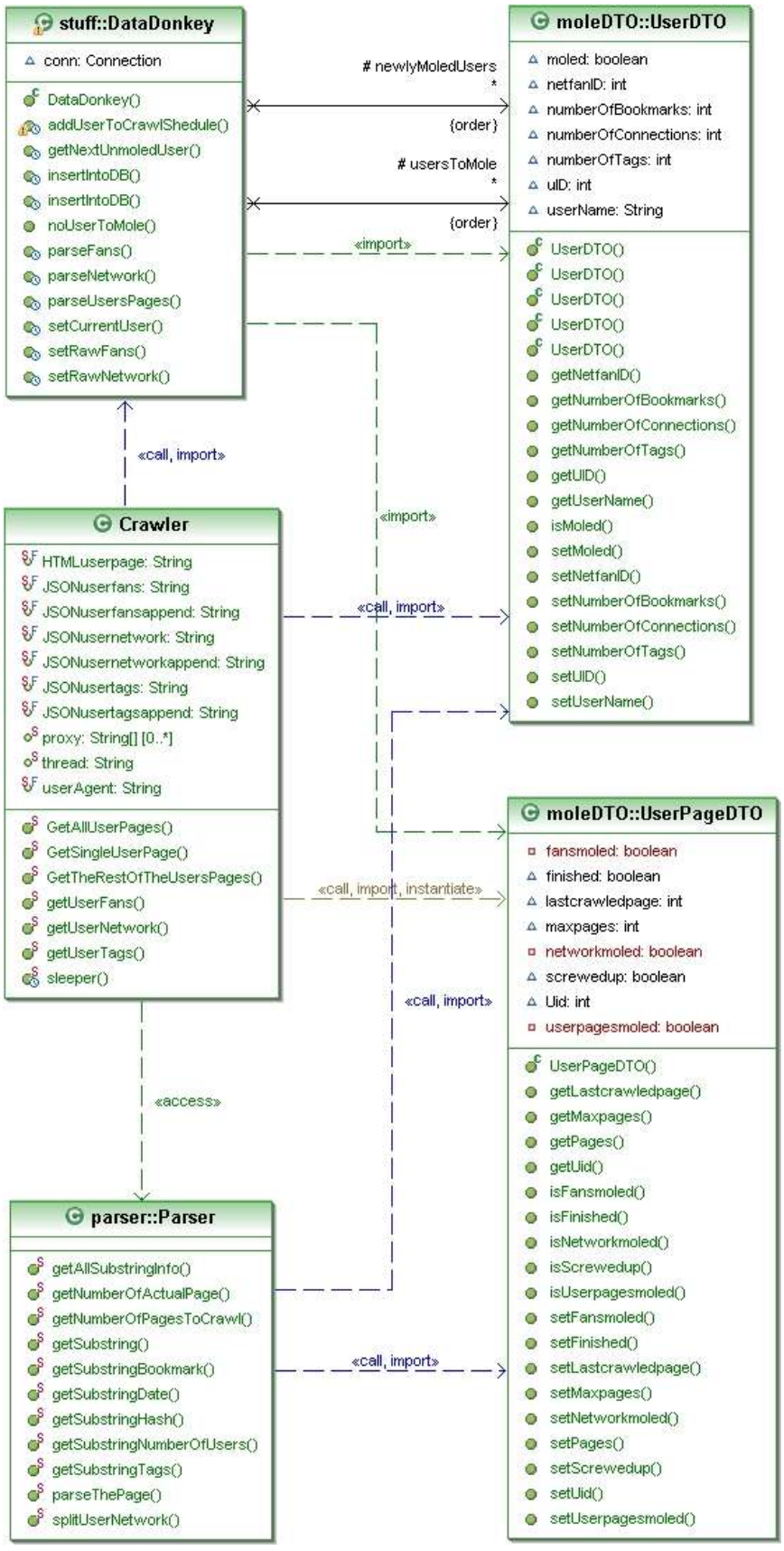
Klassendiagramm 2



Klassendiagramm 3



Klassendiagramm 4



Klassendiagramm 5

5 Datenbank

Datenbank moleicious

Tabellenstruktur für Tabelle bookmark

Feld	Typ	Null	Standard
bID	int(11)	Ja	NULL
url	varchar(333)	Ja	
url_2	varchar(5000)	Ja	NULL
numberOfUsers	int(11)	Ja	

Tabellenstruktur für Tabelle tag

Feld	Typ	Null	Standard
tID	int(11)	Ja	NULL
tagName	varchar(333)	Ja	
numberOfBookmarks	int(11)	Ja	NULL

Tabellenstruktur für Tabelle user

Feld	Typ	Null	Standard
uID	int(11)	Ja	NULL
userName	varchar(100)	Ja	
numberOfBooks	int(11)	Ja	0
numberOfTags	int(11)	Ja	0
numberOfConnections	int(11)	Ja	999999
moled	tinyint(1)	Ja	0

Tabellenstruktur für Tabelle userbooks

Feld	Typ	Null	Standard
uID	int(11)	Ja	
bID	int(11)	Ja	
tID	int(11)	Ja	NULL
date	date	Ja	0000-00-00

Tabellenstruktur für Tabelle userfan

Feld	Typ	Null	Standard
uID	int(11)	Ja	
fanID	int(11)	Ja	

Tabellenstruktur für Tabelle usernet

Feld	Typ	Null	Standard
uID	int(11)	Ja	
netID	int(11)	Ja	