Fachbereich Informatik

Bachelorarbeit

im Studiengang Informatik

Unterstützung des Benutzers bei der explorativen Graphanalyse

Johannes M. Pfeiffer

18. November 2008

Betreuer: Dipl.-Inf. Darko Obradovic

Prüfer: Prof. Dr. Andreas Dengel

2. Prüfer: Prof. Dr. Didier Stricker

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Johannes Pfeiffer

Zusammenfassung

Die vorliegende Bachelorarbeit beschäftigt sich mit der Frage, wie Benutzer besser bei der explorativen Analyse von Graphen unterstützt werden können. Dabei wird die bestehende Software „Guess“ angepasst und erweitert. Konkret wurden dazu zunächst die Benutzerschnittstelle und die Visualisierung der Software analysiert, um anschließend die daraus resultierenden Verbesserungsmöglichkeiten zu implementieren. Bei der Analyse der Benutzerschnittstelle wurden verschiedene Interaktionsstile untersucht und auf eine vollständige Implementierung überprüft. Des Weiteren wurden das Layout, sowie die Effizienz der Oberfläche betrachtet und Verbesserungsmöglichkeiten aufgezeigt. Bei der Analyse der Visualisierung wurden Methoden gesucht eine möglichst gute Übersichtlichkeit zu erreichen. Ebenso wurden Möglichkeiten gesucht die Aufmerksamkeit, durch Effekte, auf bestimmte Teilbereiche eines Graphen zu lenken. Die aus der Analyse hervorgegangenen Anforderungen wurden anschließend, durch mehrere neue oder angepasste Programmfunktionen, implementiert. Die ursprüngliche Software wurde bereits unter einer Open-Source Lizenz veröffentlicht, so dass die neuen Programmfunktionen wieder dem Projekt zugeführt wurden.

Abstract

This bachelor thesis investigates how users can be better supported with explorative graph analyses. Therefore, the existing software "Guess" was modified and expanded. In particular the user interface and the visualization were analysed and the identified improvement opportunities were implemented. Within the user interface analysis, different interaction styles were examined and reviewed for completeness. Furthermore the layout and the efficiency of the user interface were inspected. By analysing the visualization, methods were searched to improve the clarity as well as to put the attention on parts of the graph by using a new effect system. The resulting requirements from the analysis were implemented by means of several new or modified features. The original software was already published under a open-source license and so the new features were added to the project.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Graphen sind mathematische Strukturen mit deren Hilfe Modelle erzeugt werden können um Beziehungen zwischen Objekten zu beschreiben. Die Graphentheorie, dient als Grundlage weiterer Theorien oder kann herangezogen werden um theoretische sowie praktische Probleme aus unterschiedlichsten Bereichen zu lösen.

Die Graphen können dabei rein algorithmisch, quantitativ mit statistischen Mitteln sowie explorativ analysiert werden. Hier wird ausschließlich die explorative Analyse betrachtet, die gewöhnlich eine Visualisierung des Graphen erfordert. Dabei ist es üblich einen Graph durch eine Struktur aus Objekten wie Kreisen und Linien darzustellen – a priori ist einem Graph jedoch keine Darstellung zugeordnet.

Durch Nutzung entsprechender Software wird die Erzeugung von Visualisierungen stark vereinfacht. Ebenso kann die Software den Benutzer bei der Analyse von Graphen unterstützen. Es existiert bereits eine Vielzahl von Produkten und Frameworks für diesen Zweck. Wie bereits erwähnt kann die Graphentheorie für ein breites Spektrum an Themen herangezogen werden, weshalb sich auch viele der Softwareprodukte auf ein Spezialgebiet fokusieren. Als Beispiel sei hier Cytoscape (14) für den biologischen Bereich genannt. Andere, wie z.B. Pajek (3), sind allgemeiner ausgelegt, lassen sich jedoch nur schwer auf den gewünschten Anwendungsfall anpassen. Eine Anpassung von Pajek durch Implementierung der speziellen Anforderungen würde erstens immer ein komplettes Kompilieren der Software erfordern und zweitens den Zugriff auf den Quelltext voraussetzen. Insbesondere dies stellt sich als schwierig dar, da Pajek nicht unter einer Open-Source Lizenz veröffentlicht wird.

Ein weiteres Problem bei einer explorativen Analyse mit Pajek ist die Benutzerschnittstelle. Diese ist für eine interaktive Visualisierung nur sehr bedingt geeignet, da lediglich über viele einstellbare Parameter statische Bilder

des Graphen erzeugt werden können. Die starre Benutzerführung erschwert es zusätzlich, die Visualisierung kurzfristig zu ändern. der Parameter zur Erzeugung einer angepassten Version der Visualisierung.

Dieses Problem hat auch die Software UCINET (5), welche zur Bedienung fast ausschließlich auf Menüs setzt. Wie in der Bedienungsanleitung bereits angegeben, wurde die Priorität bei der Entwicklung nicht auf die Benutzerschnittstelle gelegt („[...] UCINET 5.0 is built for speed, not for comfort.“, (4)). Für eine schnelle, explorative Analyse ist die Software zu umständlich. Obwohl die Funktionalität sehr umfangreich ist, gibt es keine geeigneten Erweitungsmöglichkeiten um die Software anpassen zu können.

Für diese Zwecke besser geeignet ist die Software Guess (1). Diese wurde entwickelt um ein flexibles Werkzeug für die Analyse von Graphen zu erhalten. Durch die Einbindung eines Interpreters kann die Software über eine vollständige, standardisierte und einfache Programmiersprache, ohne erneutes Kompilieren, direkt zur Laufzeit verändert werden. In Abschnitt 2.2 wird die Software kurz beschrieben.

Diese Ausarbeitung beschäftigt sich mit der Frage wie der Benutzer bei der explorativen Analyse besser unterstützt werden kann. Dazu werden in Kapitel 2 zunächst einige Grundlagen angesprochen. Das folgende Kapitel beschreibt die Ziele, welche mit dieser Arbeit erreicht werden sollen. In den Kapiteln 4 und 5 wird die Software Guess auf Möglichkeiten zur Verbesserung analysiert und anschließend werden einige Ausschnitte der Implementierung beschrieben. Im letzten Kapitel werden diese Veränderungen beurteilt und es wird ein Ausblick auf weitere Möglichkeiten gegeben.

Kapitel 2

Grundlagen

In diesem Kapitel sollen zunächst einige theoretische Grundlagen der Graphentheorie erläutert werden. Dabei werden weitestgehend die Notationen und Definitionen aus (7) verwendet. Weiterhin werden die Funktionen und der grobe technische Aufbau von Guess erläutert um die später in den Kapiteln 4 und 5 beschriebenen Arbeiten einzuordnen.

2.1 Theoretische Grundlagen

2.1.1 Elementare Begriffe

Ein **Graph** G ist definiert durch das Paar $G = (V, E)$, mit $E \subseteq V^2$. Dabei bezeichnet V die Menge der Knoten (von engl. vertex) und E die Menge der Kanten (von engl. edge). Im Gegensatz zu diesem **ungerichteten Graph** hat ein **gerichteter Graph** zusätzlich zwei Funktionen $init : E \to V$ und $ter : E \to V$. Dabei weist $init$ jeder Kante e den Anfangsknoten $init(e)$ zu und ter weist jeder Kante e den Endknoten $ter(e)$ zu. Gilt $init(e) = ter(e)$, so ist die Kante e eine **Schlinge**.

Sei $G = (W, F)$. $V(G)$ bezeichnet die Knotenmenge W, $E(G)$ die Kantenmenge F. Die **Ordnung** von G wird als $|G|$ bezeichnet, was dabei eine andere Schreibweise für $|V(G)|$ ist.

Die Kante $\{v_1, v_2\} \in E(G)$ wird durch v_1v_2 abgekürzt. Ein Knoten v und eine Kante e heißen **inzident**, wenn $v \in e$. Zwei Knoten v_1, v_2 sind **adjazent** in G, wenn die Kante $v_1v_2 \in E(G)$. Sind alle Knoten in G paarweise adjazent, so heißt G **vollständig** und wird mit $K^{|G|}$ bezeichnet.

Das **Komplement** $\overline{G}$ bezeichnet den Graphen in dem zwei Knoten genau dann adjazent sind, wenn sie es in G nicht sind. Der **Grad** $d(v)$ eines Knotens v ist die Anzahl der zu v inzidenten Kanten.

Ein **Weg** ist ein nicht leerer Graph W der Form $W = (V_w, E_w)$ mit $V_w = \{v_0, v_1, ..., v_k\}$ und $E_w = \{v_0v_1, v_1v_2, ..., v_{k-1}v_k\}$, wobei die v_i paarweise verschieden sind. Seine Länge ist durch die Anzahl der Kanten festgelegt.

Sei G ein Graph mit $G = (V, E)$ und G' ein weiterer Graph mit $G' = (V', E')$. G heißt **isomorph** zu G', also $G \simeq G'$, wenn es eine Bijektion $\varphi : V \to V'$ gibt mit $\{v_1, v_2\} \in E \Leftrightarrow \{\varphi(v_1), \varphi(v_2)\} \in E'$ für alle $v_1, v_2 \in V$. Zwischen zueinander isomorphen Graphen wird hier nicht unterschieden und anstatt $G \simeq G'$ wird $G = G'$ geschrieben.

Gilt $V' \subseteq V$ und $E' \subseteq E$, so ist G' ein **Teilgraph** von G, mit der Notation $G' \subseteq G$ oder auch $G[G']$. G' heißt **induziert**, wenn er alle Kanten $\{v_1, v_2\} \in E$ mit $v_1, v_2 \in V'$ enthält. G' wird dann auch **Untergraph** genannt und auch mit $G[V']$ bezeichnet.

2.1.2 Auswahl von Teilgraphen

Es gibt eine Reihe von Möglichkeiten um geeignete Teilgraphen eines Graphen zu finden. Die Motivation ist dabei das Finden von Zusammenhängen, das Gruppieren von Knoten bzw. Kanten mit ähnlichen Eigenschaften oder eine ästhetisch ansprechendere Anordnung.

Häufig werden den Knoten und Kanten in der Analyse weitere Attribute oder Eigenschaften hinzugefügt um ein geeignetes Modell des untersuchten Problems zu erzeugen. Hier soll jedoch die Auswahl der Teilgraphen zunächst nur auf Basis der Struktur des Graphen erfolgen. Im den folgenden Abschnitten werden zwei Verfahren betrachtet, die wie in (13) als Komponenten und Cliquen bezeichnet werden.

Komponenten

Das Konzept der Komponenten ist relativ simpel: Erzeuge einen Teilgraph mit der größtmöglichen Knotenmenge V_K, wenn jeder Knoten $v_1 \in V_K$ mit jedem anderen Knoten $v_2 \in V_K$ des Teilgraphen über einen Weg verbunden ist. D.h. in einer Komponente gibt es von jedem Knoten einen Weg zu jedem anderen Knoten. Ein vollständiger Graph besteht trivialerweise nur aus einer Komponente, ein isolierter Knoten kann zu keiner Komponente gehören (siehe Abbildung 2.1).

Im Unterschied zu den **einfachen Komponenten** bei ungerichteten Graphen wird bei gerichteten Graphen zwischen **starken Komponenten** und **schwachen Komponenten** unterschieden. Bei starken Komponenten fordert man, dass die Richtung der Kanten in einem Weg von einem Knoten zu einem beliebigen anderen Knoten gleich bleibt. Dadurch wird z.B. der Fluss einer Ressource modelliert.

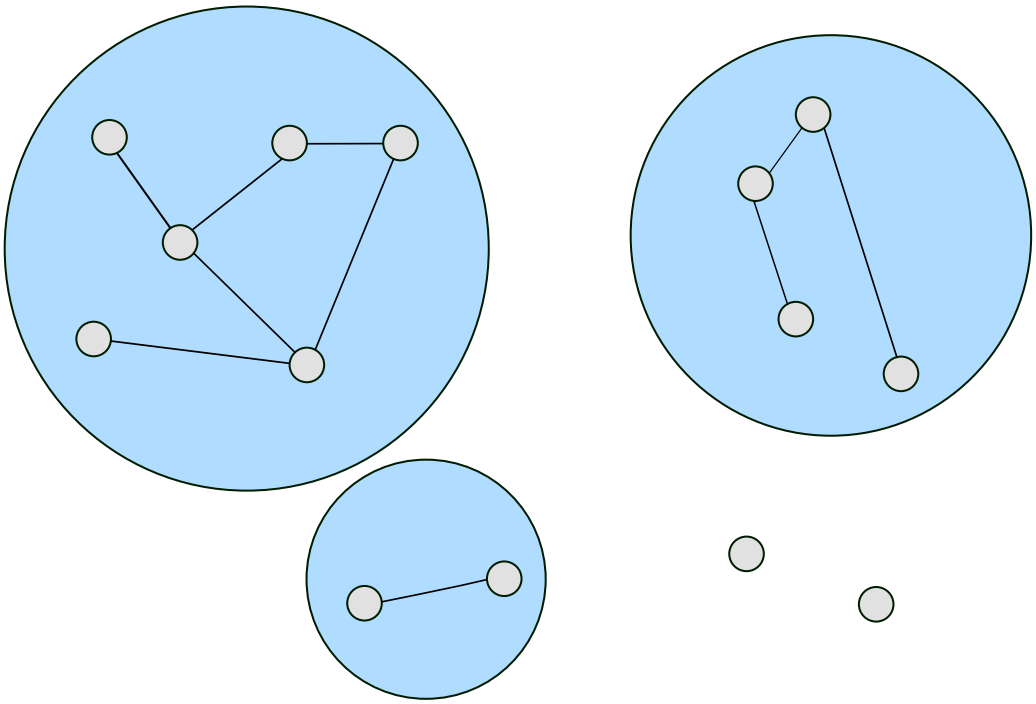

Abbildung 2.1: Darstellung eines Graphen nach (13) mit drei Komponenten und zwei isolierten Knoten.

Schwache Komponenten müssen diese Forderung nicht erfüllen - in einem Weg darf sich die Richtung der Kanten ändern. Es geht bei einer schwachen Komponente also mehr um die Existenz einer Beziehung zwischen den Knoten, als um den Fluss einer Ressource.

Das Resultat ist eine Visualisierung mit einer oder mehreren einfachen, starken bzw. schwachen Komponenten und evtl. isolierten Knoten. Die Größe und Anzahl der Komponenten ist interessant für eine weitere Analyse.

Cliquen

Cliquen sind ähnlich zu Komponenten. Bei Komponenten wird nur gefordert, dass die Knotenmenge maximal und durch Wege verbunden ist. Bei Cliquen wird gefordert, dass die Knotenmenge maximal ist und der Teilgraph vollständig ist, d.h. alle Knoten sind adjazent zueinander. Man kann hier analog von starken und schwachen Cliquen sprechen, stark dann, wenn jede ausgehende Kante einen wechselseitigen Partner besitzt.

Dieses Konzept hat sich jedoch als zu restriktiv herausgestellt, weswegen mehrere Erweiterungen entwickelt wurden. Eine davon ist die **n-Clique**: n gibt dabei die maximale Länge des Weges an den zwei Knoten entfernt sein

dürfen. Eine 1-Clique entspricht der ursprünglichen Definition. In einer 2-Clique sind alle Knoten entweder direkt verbunden (über genau eine Kante) oder über einen weiteren Knoten verbunden (über genau zwei Kanten).

2.1.3 Layouts

Beim Zeichnen der Graphen wird durch so genannte Layouts ein Algorithmus angegeben der die Darstellung des Graphen angibt. Dieser muss sich nicht nur an der Struktur des Graphen orientieren, sondern kann auch die weiteren Attribute der Knoten und Kanten verwenden. In Guess wird der Graph dazu auf eine euklidische Ebene abgebildet indem die Knoten i.A. als Kreis und die Kanten als Verbindungsgeraden gezeichnet werden. Über die X- und Y-Position der Knoten kann deren Position in der Ebene bestimmt werden, wodurch die Position der Kanten automatisch angepasst wird.

In Guess können die Layouts momentan zwar nur die Knotenpositionen verändern, allgemein kann beim Zeichnen von Graphen aber auch die Art der Kanten geändert werden. In (12) werden dazu verschiedene Kantentypen wie „polyline-“, „orthogonal-“, „grid-based-“ und „straight-line-drawing“ vorgestellt, wobei nur letzterer Typ in Guess verwendet wird.

Zwei der in Guess verfügbaren Layouts sollen hier kurz vorgestellt werden.

Fruchterman-Rheingold

Das Fruchterman-Rheingold Layout (8) basiert auf Anziehungs- und Abstoßungskräften von Knoten, wobei sich alle Knoten abstoßen. Sind die Knoten adjazent, erhalten diese pro gemeinsamer Kante zusätzliche Anziehungskraft. Ansonsten werden die Knoten gleichmäßig auf die Zeichenfläche verteilt.

Circular

Das Circle Layout verschiebt die Knoten auf den Rand einen Kreises (siehe Abbildung 2.2). Dazu wird zuerst der Mittelpunkt des bisherigen Graphen berechnet indem das arithmetische Mittel aus den X- bzw. Y-Werten der am weitesten voneinander entfernten Knoten in X bzw. Y-Richtung gebildet wird. Der *Radius* wird auf die Hälfte von $max\{Höhe, Breite\}$ des von den Knoten aufgespannten Bereichs gesetzt. Für jeden Knoten i der insgesamt n Knoten wird die X- bzw. Y-Position durch folgende Formeln festgelegt:

$$x_i = Radius * \cos(2 * \pi * \frac{i}{n}) + x_{Mittelpunkt}$$

$$y_i = Radius * \sin(2 * \pi * \frac{i}{n}) + y_{Mittelpunkt}$$

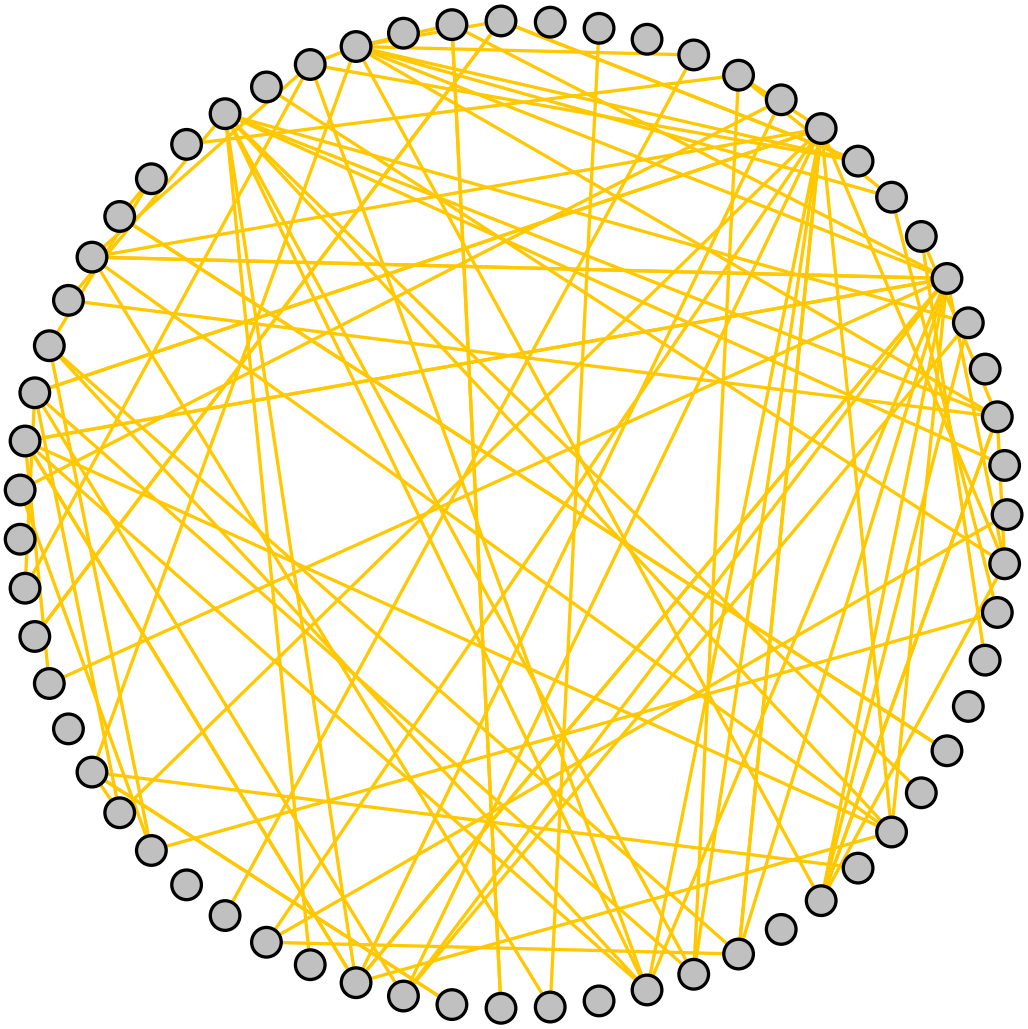

Abbildung 2.2: Circular Layout mit 63 Knoten und 102 Kanten.

2.2 Technische Grundlagen

2.2.1 Überblick der Funktionen

Guess wurde entwickelt um eine Software zu erhalten die möglichst einfach an die Bedürfnisse des jeweiligen Anwendungsbereichs anpassbar ist. Es entstand aus Zoomgraph (2), einem Projekt der HP Laboratories. Es wird bereits in einem breiten Feld von Anwendungsbereichen eingesetzt (1).

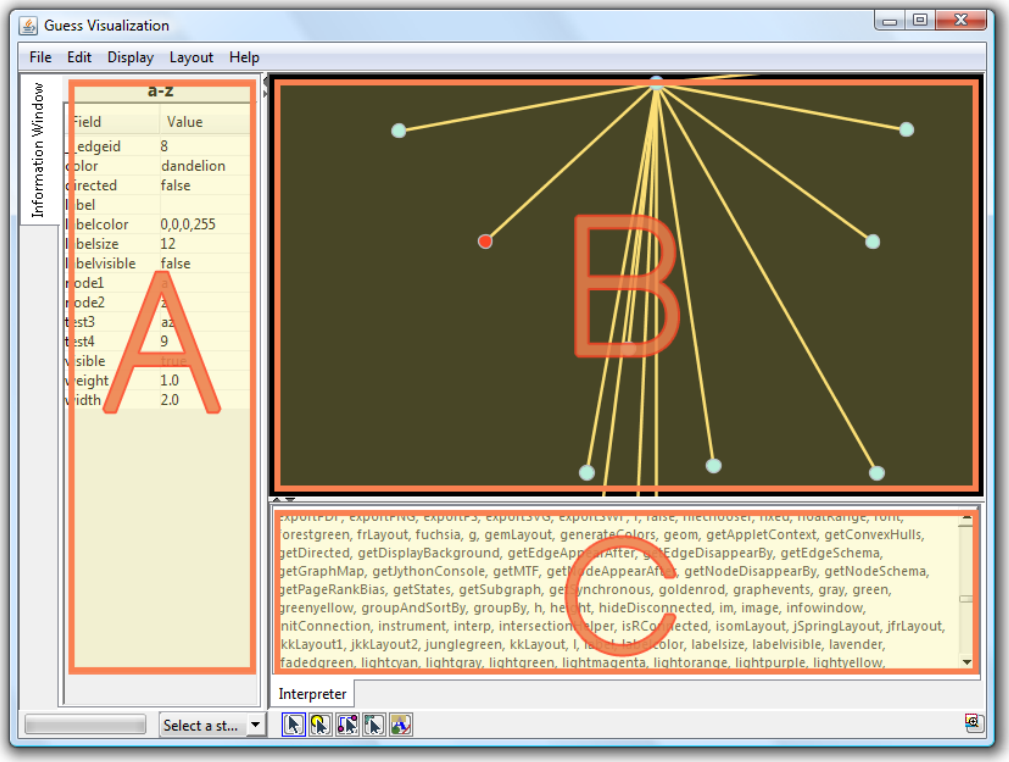

Abbildung 2.3: Hauptfenster Guess mit (A) Information Window, (B) Visualisierung und (C) Konsole mit Gython Interpreter.

Visualisierung

Guess ermöglicht das Erzeugen und Exportieren von statischen Visualisierungen, wie Bildern, und dynamischen Visualisierungen in Form von Videos. Dabei kann die Visualisierung während der Arbeit mit Guess auch interaktiv genutzt werden.

Knoten und Kanten lassen sich in einem speziellen Modus mit der Maus in der Position und Größe verändern. Weitere Eigenschaften lassen sich über die Konsole (siehe Abbildung 2.3 C), den Feld Editor (siehe Abbildung 4.11) oder das Information Window (siehe Abbildung 2.3 A) anpassen.

Um Graphen in eine bestimmte Darstellung zu bringen kann der Benutzer auch auf unterschiedliche Layouts zurückgreifen (siehe Fruchterman-Rheingold und Circle in Abschnitt 2.1.3).

Interaktiver Interpreter

Durch die Scriptsprache Gython, welche auf Jython[1] aufbaut, können sehr einfach und direkt Veränderungen an dem Graph, dessen Visualisierung oder auch an Guess selbst vorgenommen werden. Durch eine objekt-orientierte Syntax werden Methoden aufgerufen oder Attribute von Objekten manipuliert.

Um die Bedienung zu erleichtern wurde der Zugriff auf die häufig benötigten Objekte vereinfacht. Der Graph wird durch das Objekt `g` angesprochen, seine Knoten und Kanten durch `g.nodes` bzw. `g.edges`. Es existieren Shortcuts um die Auswahl bestimmter Objekte zu vereinfachen: So wird das Objekt der (gerichteten) Kante von den Knoten V1 nach V2 durch `(V1->V2)` erzeugt. Dessen Attribute oder auch Felder können so direkt bearbeitet werden, so setzt beispielsweise `(V1->V2).color = green` die Farbe der entsprechenden Kante auf grün. Knoten und Kanten können beliebig mit Feldern versehen werden, wobei ein Feld aus einer Bezeichnung und einem Typ (Text, Zahl oder Boolean) besteht. Es gibt visuelle Felder (wie im obigen Beispiel color), deren Änderung eine Auswirkung auf die Visualisierung hat und einfache Datenfelder um z.B. Knoten bestimmte Attribute zuweisen zu können. Daneben existieren auch Felder wie „indegree“, „outdegree“ oder „totaldegree“, die dynamisch berechnet werden und auf die nur ein Lese-Zugriff besteht.

Um den Umgang mit mehreren Objekten zu vereinfachen wurde eine Mengen- bzw. Gruppenverwaltung implementiert: `gruppe1 = (V1,V2,V3)` erzeugt eine Referenz auf die Menge von Knoten $\{V1, V2, V3\}$. Ein Zugriff auf die Felder der Gruppe wirkt sich wie ein Zugriff auf die Objekte selbst aus. Durch die Operatoren „&“ und „|“ kann aus mehreren Gruppen eine neue Gruppe aus der Vereinigungs- bzw. Schnittmenge gebildet werden.

Eine weitere Möglichkeit Knoten und Kanten anzusprechen ist die Möglichkeit diese nach Feldern zu filtern. Dazu wird eine Query-ähnliche Syntax verwendet. Der Befehl

```
((y > 500) & (color == blue)).color = green
```

setzt die Farbe aller Objekte mit der Farbe Blau auf die Farbe Grün, falls ihre Y-Position größer als 500 ist.

Dieser Ausbau der Skriptsprache ermöglicht eine direkte und intuitive Analyse des Graphs. Komplexere Skripte lassen sich auch aus einer Datei laden.

[1] Jython ist eine Java Implementierung von Python.

Zustände

Zustände wurden aus zwei Gründen implementiert: Zum Ersten, um die Visualisierung zu einem früheren Zeitpunkt wiederherstellen zu können. Das ist den Benutzern einer Umfrage aus (1) sehr wichtig. Hat beispielsweise das Anwenden eines Layouts nicht das gewünschte Ergebnis erbracht, kann man wieder zu einem vorherigen Zustand zurückkehren. Dabei wurde allerdings keine Undo / Redo Funktionalität implementiert (siehe Analyse in Kapitel 4). Ein Zustand muss vorher über `ss("<Bezeichnung>")` gespeichert werden um später über `ls("<Bezeichnung>")` geladen werden zu können.

Der zweite Grund für den Einsatz der Zustände ist die Möglichkeit auf einfachem Weg eine dynamische Visualisierung erzeugen zu können. Zwischen zwei Zuständen kann mit dem Befehl `morph` eine Animation erzeugt werden. Dabei dienen jeweils zwei Zustände als die Schlüsselbilder zwischen denen Zwischenbilder berechnet werden.

2.2.2 Architektur

Dieser Abschnitt soll einen Überblick über den Aufbau von Guess geben. In Abbildung 2.4 ist eine Einteilung in die verschiedenen Komponenten und verwendeten Bibliotheken dargestellt. Folgend wird die Funktion und der Aufbau der Komponenten beschrieben.

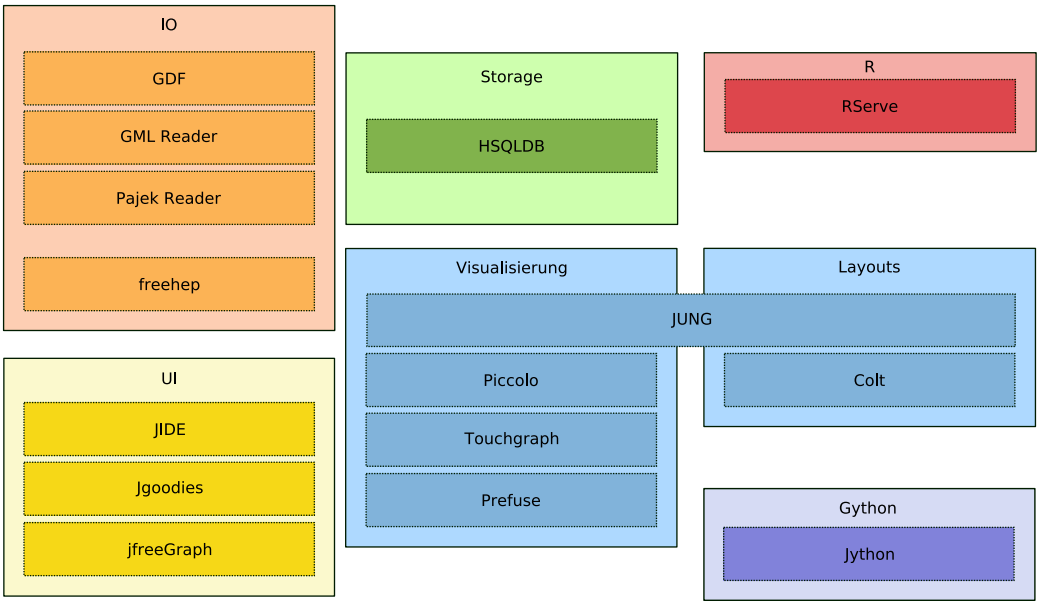

Abbildung 2.4: Übersicht der Architektur und eingebundenen Bibliotheken.

IO

Die Ein- und Ausgabe importiert bzw. exportiert Graphdaten. Das eigene Format GDF kann geöffnet und gespeichert werden. Dabei werden jedoch nur die Daten der Knoten und Kanten mit den jeweiligen Attributen verarbeitet. Weitere Informationen, wie z.B. Annotationen, gehen beim Speichern in GDF verloren.

Zusätzlich stehen Importfunktionen für GraphML und Pajek zur Verfügung. Beim Einlesen der Daten werden die entsprechenden Knoten- und Kanten-Objekte erstellt und entsprechende Variablen für Gython erstellt.

Per freehep können Visualisierungen als Bild exportiert werden. Dazu stehen die Formate JPEG, PNG, PPM, RAW, CGM, EMF, PDF, EPS, PS, SVG, SWF und GIF zur Verfügung. Die Erzeugung von Videos im MOV Format reiht lediglich mehrere JPEG-Bilder hintereinander. Dies wird über das Java Media Framework durchgeführt.

UI

Die Benutzerschnittstelle hält alle Dialoge und Steuerelemente bereit. Über das JIDE Toolkit werden u.a. die Docking Funktionen für Konsole und Information Window bereitgestellt. Die Jgoodies Bibliothek bietet Steuerelemente an, welche in Guess genutzt werden können. Die Bibliothek jfreeGraph enthält Funktionen zum Zeichnen von Diagrammen und dazugehörigen Legenden.

Storage

Die Speicherung der Daten wird abstrahiert, so dass diese z.B. in einem eingebetteten SQL-Server (wie momentan HSQLDB) oder in einer anderen Implementierung der entsprechenden Schnittstellen gespeichert werden können.

R

Per RServe kann zu einem RServer verbunden werden, um statistische Berechnungen durchzuführen. Bei der explorativen Graphanalyse spielt dies jedoch eine untergeordnete Rolle.

Visualisierung

Die Visualisierung wurde so flexibel gestaltet, dass das ZUI[2] ausgetauscht werden kann. Als Beispiel wurden Piccolo, Touchgraph und Prefuse implementiert. JUNG nimmt eine Sonderrolle ein, da hier nicht nur Funktionen zur Visualisierung von Graphen sondern auch Funktionen zu Verwaltung von Graphen vorhanden sind. Standardmäßig aktiv ist die Piccolo-Visualisierung.

Layouts

JUNG enthält neben dem grundlegenden Graphframework auch einige der Funktionen um Layouts auf den Graphen anzuwenden. Colt ist eine Bibliothek zur Bereitstellung von wissenschaftlichen und technischen Funktionen, welche von einigen Layouts benötigt werden.

Gython

Gython basiert wie schon in Abschnitt 2.2 erwähnt auf Jython und wurde um einige Features zum vereinfachten Zugriff auf Graph-Objekte wie Knoten und Kanten erweitert.

[2]Zoomable User Interface

Kapitel 3

Zielsetzung

Das Ziel dieser Arbeit ist eine bessere Unterstützung des Benutzers bei der Analyse von Graphen. Dabei sollen an dem Programm Guess Modifikationen und Erweiterungen vorgenommen werden, um zum einen die Benutzerschnittstelle, und zum anderen die Visualisierung und Präsentationsmöglichkeiten des Graphen zu verbessern.

Die gewünschte Benutzerschnittstelle soll dabei aus einer einfachen, konsistenten und effizienten Oberfläche bestehen, welche sich an eine wissenschaftliche Zielgruppe richtet. Die Änderungen zur Visualisierung des Graphen zielen auf mehr Übersichtlichkeit ab. Dazu werden u.a. ein neues Zoom-Verfahren und ein neuer Layout-Algorithmus implementiert. Schließlich soll der Benutzer auch bei der Erzeugung von Videos für Präsentationen oder zum Austausch mit anderen Interessierten besser unterstützt werden.

Die praktischen Ergebnisse der Arbeit werden dem Open-Source Projekt Guess und damit der Allgemeinheit wieder zur Verfügung gestellt. Aus diesem Zweck werden die Änderungen am Quellcode im CVS des Projekts (16) und als CVS-Patches in der Mailingliste veröffentlicht.

Kapitel 4

Analyse

In diesem Kapitel werden die Benutzerschnittstelle, die Visualisierung und Präsentation von Guess untersucht. Es werden grob der ursprüngliche Stand der Software und die dabei auftretenden Probleme aufgeführt. Dabei werden die Verbesserungsmöglichkeiten, welche im Rahmen dieser Arbeit umgesetzt wurden, aufgezeigt. Nicht extra aufgeführt werden einige durchgeführte Fehlerkorrekturen an Guess oder den verwendeten Bibliotheken.

4.1 Benutzerschnittstelle

4.1.1 Interaktion

Die Benutzerschnittstelle von Guess verwendet mehrere Interaktionsstile, wie in Abbildung 4.1 zu sehen. Die Einteilung in die verschiedenen Stile erfolgt nach (15). Es werden die vier Interaktionsstile „Direct Manipulation“, „Menu Selection“, „Dialog Boxes“ und „Command Language“ verwendet.

Direct Manipulation

Guess setzt „Direct Manipulation“ ein um Aktionen zur Navigation durch den Graph (per Pan & Zoom), zum Anpassen der Größe und Position von Knoten, Kanten und Hüllen, sowie zum Erzeugen von Annotationen umzusetzen. Dieser Interaktionsstil wird dabei durch die folgenden Eigenschaften charakterisiert (siehe auch (15)):

1. Die Knoten, Kanten und relevanten Aktionen werden permanent angezeigt. Die verfügbaren Aktionen und die aktuell ausgewählte Aktion werden dabei durch visuelle Metaphern in Form von Symbolen in der Statusleiste angezeigt.

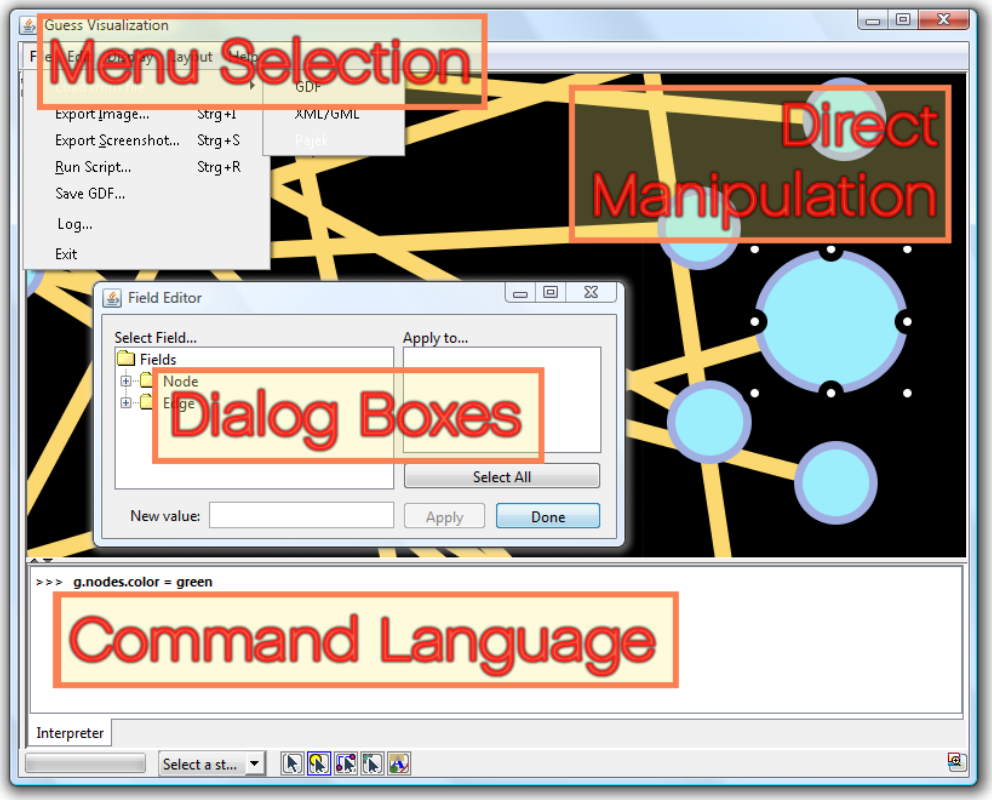

Abbildung 4.1: Beispiele für den Einsatz der verwendeten Interaktionsstile im Hauptfenster.

2. Aufgaben, wie die Positionsänderung eines Knotens, werden durch physische Aktionen, in diesem Fall durch Ziehen des Knotens mit der Maus, durchgeführt.

3. Schnelle, inkrementelle und reversible Aktionen, deren Effekt auf die Objekte unmittelbar sichtbar wird. Schnell und inkrementell sind die angesprochenen Aktionen offensichtlich, da z.B. die Größenänderung eines Knotens unmittelbar angezeigt wird und nachträglich verfeinert werden kann. Automatisch reversibel sind die Aktionen jedoch nicht. Lediglich über den Umweg, explizit einen Zustand vor dem Ausführen der Aktion anzulegen, kann später die Aktion rückgängig gemacht werden. Eine Funktion zum rückgängig machen von Aktionen ist sehr wünschenswert, da Benutzer dadurch auch ein verringertes „Angstgefühl“ beim Erkunden von unbekannten Funktionen empfinden können (15). Daraus folgt, dass der Benutzer das Programm nach mehr, für ihn

neuen, Funktionen erkundet und es dadurch besser zu bedienen lernt. Deshalb wurde eine Rückgängig- / Wiederholen-Funktion implementiert.

Menu Selection

Die „Menu Selection“ findet an mehreren Stellen im Programm statt: Einmal wie in Abbildung 4.1 zu sehen im Hauptmenü, zusätzlich auch in den Kontextmenüs der Knoten und Kanten. Das Hauptziel von Menüs besteht allgemein darin, eine vernünftige, verständliche, merkbare und zweckmäßige Organisation der Aufgaben des Benutzers zu erstellen (15). Dabei sind die Bezeichnungen, die Gruppierung und Anordnung der Menüeinträge Möglichkeiten Einfluss auf dieses Ziel zu nehmen.

Bei den Bezeichnungen sollte eine Konsistenz innerhalb des Programms, aber auch allgemein akzeptierte Konventionen eingehalten werden. Ein Beispiel für eine allgemein akzeptierte Konvention ist die Ellipse am Ende einer Bezeichnung. Sie enthält einen Hinweis darauf, dass dort eine Benutzereingabe erwartet wird. Oder in Kontextmenüs wird die Standard-Aktion fett dargestellt (18). Gleichzeitig sollen die Bezeichnungen untereinander auch ausreichend voneinander unterscheidbar sein, so dass keine Missverständnisse auftreten.

Die Gruppierung sollte so flach wie möglich sein um den Benutzer nicht unnötig lange in vielen Untermenüs suchen zu lassen und die Aktionen sollten sinnvoll gruppiert werden, z.B. nach Aufgabenart.

Die Anordnung der Elemente sollte ebenfalls nicht zufällig, sondern nach klaren Regeln vorgenommen werden. Beispielsweise können die Einträge nach Ähnlichkeit, Alphabet oder Wichtigkeit sortiert werden.

Es gibt einige Kritikpunkte an der Umsetzung der „Menu Selection“ in Guess. Bei den Konventionen zur Bezeichnung hat z.B. im Kontextmenü der Eintrag „Center On...“ eine Ellipse obwohl keine Benutzereingabe erwartet wird. Umgekehrt hat der Eintrag „Radial“ unter dem Menüpunkt „Layouts“ keine Ellipse, ebenso wie der Menüeintrag „Background Color“ im Menü „Display“, obwohl dort Benutzereingaben erwartet werden. Einige Einträge wie „Export Image...“ und „Export Screenshot...“ lassen sich aufgrund der Bezeichnung nicht genügend unterscheiden um auf die verschiedenen Funktionen zu schließen. Einige Bezeichnungen entsprechen nicht den allgemeinen Konventionen, so wird „Display“ in der Regel mit „View“ bezeichnet (18). Ebenfalls entgegen der allgemeinen Konventionen wird der erste Menüeintrag im Kontextmenü als eine Art Titel verwendet und ist ansonsten nicht benutzbar. Das verwirrt insofern, als dass sich der Titel vom Aussehen und Verhalten nicht von den anderen Menüeinträgen unterscheidet. Der Stil der

Menüeinträge wird nicht konsequent in der Headline Capitalization umgesetzt. Aus diesem Grund wurden die Bezeichnungen der Menüs komplett überarbeitet und entsprechen nun den Konventionen.

Zur Gruppierung der Elemente kann man anmerken, dass „Toogle Arrows“ und „Background Color“ besser zu den Elementen aus „Edit“ als zu denen aus „Display“ passen, da hier eine Änderung vorgenommen wird und nicht nur die Anzeige angepasst wird. Die Unterteilung im Menü „Import from file“ ist nicht besonders hilfreich, wird doch bei jedem Untermenüpunkt (GDF, XML/GML, Pajek) der gleiche Dialog zur Auswahl einer Datei angezeigt. Deshalb wird dieses Untermenü entfernt und ein einzelner Menüpunkt „Import from File...“ angelegt. Der Menüpunkt „Radial“ unter „Layout“ erwartet als Eingabe die entsprechende Bezeichnung eines Knoten auf den das Layout angewendet wird. Dabei muss sich der Benutzer den Namen des Knotens merken, bevor die Aktion ausgewählt wird. Besser ist es die Aktion zusätzlich ins Kontextmenü der Knoten aufzunehmen. Bei der Anordung innerhalb des Menü „Layout“ scheint ebenfalls keine Regel erkennbar. Eine beispielsweise alphabetisch geordnete Liste könnte hier von Vorteil sein. Um die Befehle besser im Gedächtnis des Benutzers zu behalten wurden die meisten Menüeinträge mit Symbolen versehen. So kann auch teilweise auf das erneute Durchlesen der Menüs verzichtet werden, da bereits das Symbol erkannt wurde. In den Menüs werden dabei die Symbole des Tango Desktop Projekts (19) verwendet. Diese bieten eine hohe Qualität, sind gut unterscheidbar, lassen sich leicht erkennen und wurden unter einer Open-Source Lizenz (CC BY-SA) veröffentlicht.

Dialog Boxes

Die „Dialog Boxes“ (eine Erweiterung des „Form-Fillin“ Interaktionsstils (15)) werden angewendet, wenn eine Aktion über „Menu Selection“ nicht mehr zweckmäßig ist, da z.B. die Eingabe von Text erwartet wird. Dabei sollte wieder auf Konsistenz zwischen den Dialogen geachtet werden, was aber weder im Layout, noch mit den Bezeichnungen konsequent umgesetzt wurde. Einige Dialoge lassen auch eine Beschreibung der einzelnen Felder vermissen, so dass deren Bedeutung nur erraten werden kann. Andere Dialoge (wie der „Field Editor“) lassen eine Möglichkeit zum Abbrechen vermissen. Alle Dialoge werden deshalb umfassend überarbeitet bzw. neu geschrieben. Auch wird nun auf die korrekten Standard-Dialoge zurückgegriffen, d.h. bei der Auswahl von Ordnern wird nun ein Ordner-Dialog verwendet und beim Speichern von Dateien ein Speicher-Dialog.

Command Language

Der letzte verwendete Interaktionsstil wird als „Command Language“ bezeichnet. Die Konsole stellt dabei das mächtigste Werkzeug in Guess dar, da damit jede mögliche Aktion in Guess ausgeführt werden kann. Auch komplexere Aufgaben können mit relativ wenig Befehlen ausgeführt werden. Die Geschwindigkeit mit der eine komplexe Aufgabe eingegeben wird ist der Auswahl eines Menüeintrags oder dem Ausfüllen eines Dialogs meist überlegen. Der offensichtliche Nachteil ist jedoch, dass die Befehle erst erlernt und auch behalten werden müssen. Das Erlernen wird durch eine einfache Syntax erleichtert, eine Referenz der verfügbaren Befehle wäre jedoch hilfreich. Im Menü „Help“ wurde deshalb ein Link zur Wiki-Seite von Guess eingebaut, um Befehle nachschlagen zu können.

Bei einfachen und evtl. selten genutzten Befehlen ist der Nutzen der Konsole eher gering, da sich der Benutzer erst an den entsprechenden Befehl wie z.B. für Vollbildschirm, Ein/Ausblenden der Konsole etc. erinnern muss, oder ihm der Befehl gar nicht erst bekannt ist. Da ein Menüeintrag hier offensichtlich Vorteile bringt wurden die Menüeinträge „Fullscreen“, „Console“ und „Information Window“ ins „View“ Menü aufgenommen.

4.1.2 Effizienz

Um den Benutzer bei der Arbeit entsprechend zu unterstützen sollte die Bedienung der Software effizient sein. D.h. der Benutzer soll sich mit der Analyse des Graphen befassen können und nicht über die Bedienung der Software nachdenken. Häufig wiederkehrende Aufgaben sollten deshalb erleichtert werden indem z.B. die beim letzten Aufruf getroffene Wahl gespeichert wird und die Anzahl der wiederkehrenden Dialoge gering gehalten wird. Dies wird in Guess jedoch nicht umgesetzt. Jeder Start erfordert das Ausfüllen zweier Dialoge bevor das Hauptfenster mit dem Graph erscheint. Dabei müssen die Angaben jedes Mal erneut eingegeben werden, obwohl sich meist nichts oder nur wenig ändert. Auch der Dialog zur Auswahl von Dateien z.B. zum Import oder Export merkt sich den letzten Pfad nicht. Hier muss u.U. erst jedes Mal umständlich im Dateisystem navigiert werden. Durch das Merken von bereits ausgefüllten Werten kann die Effizienz der Software gesteigert werden. Hier sind nicht nur die Textfelder gemeint, sondern auch die Hintergrundfarbe, die Fenstergröße und Position etc. werden gespeichert. Um alle gespeicherten Werte auf die Standardwerte zurückzusetzen wurde der Kommandozeilenparameter `--reset` hinzugefügt.

Ebenso trägt auch eine geringere Anzahl von Dialogen, bei gleich bleibender Übersichtlichkeit, zu mehr Effizienz bei. Dies wurde in einem neuen

Start-Dialog verwirklicht, der nun alle vorherigen Fenster in einem Dialog zusammenfasst (siehe Abbildung 4.10). Wird nun in diesem Dialog noch die Ok-Schaltfläche als Standard-Schaltfläche gesetzt (und damit mit der Return-Taste verbunden) kann der Start auf das Drücken der Return-Taste reduziert werden.

Wie im vorherigen Abschnitt zur Interaktion besprochen werden relativ viele unterschiedliche Interaktionsstile verwendet. Dabei kostet ein Wechsel zwischen den Stilen Zeit, besonders wenn zwischen der Maus und Tastatur gewechselt wird. Wie aus Untersuchungen zu dem GOMS-Modell[1](6) oder der vereinfachten Version, dem Keystroke-Level Model, hervorgeht, ist es sinnvoll bei Tastatureingaben nicht immer wieder für bestimmte Aktionen auf die Maus zurückgreifen zu müssen und umgekehrt. D.h. das Menü und die Dialoge sollten so angepasst werden, dass auch nur mit der Tastatur jedes Feld erreicht werden kann. Das kann relativ einfach über so genannte Mnemoniks implementiert werden. Dabei wird dem Feld der Fokus übergeben, falls der Benutzer einen (meist unterstrichenen dargestellten) Buchstaben in Zusammenhang mit einer weiteren speziellen Taste drückt. Im Unterschied zu Mnemoniks ist für noch häufiger verwendete Aktionen ein eigenes Tastenkürzel sinnvoll, dessen Eingabe die Aktion direkt aufruft.

Jedoch lassen sich nicht alle Aktionen sinnvoll mit der Tastatur erledigen. Gerade die Navigation im Graph oder die Manipulation von Knoten und Kanten kann über die Maus schneller erledigt werden. Eine Vorhersage über die durchschnittliche Geschwindigkeit T, mit der Benutzer die Maus in ein Ziel bewegen können, kann mit dem „Gesetz“ nach Fitts (siehe Fitts's Law) berechnet werden. Es lautet:

$$T = a + b \log_2 \frac{D}{W} + 1.$$

Dabei bezeichnen a und b zwei Konstanten die empirisch bestimmt wurden. D ist die Distanz zwischen dem Startpunkt und der Mitte des Ziels; W bezeichnet die Breite des Ziels auf der Gerade in Richtung der Bewegung. Um die Zeit T zu verkürzen kann man entweder die Distanz verringern oder die Breite des Ziels vergrößern. Zunächst wird Ersteres betrachtet: In der ursprünglichen Version wird das Übersichtsfenster (siehe Abbildung 4.2) durch einen kleinen Button in der Statusleiste (siehe Abbildung 4.4) geöffnet. Die Strecke die dazu zurückgelegt werden muss kostet relativ viel Zeit. Andererseits könnte man durch Drücken der (bis jetzt ungenutzten) mittleren Maustaste das Übersichtsfenster an der aktuellen Cursorposition öffnen. Was einer Distanz von Null entsprechen würde und damit T minimiert.

[1]Goals, Operators, Methods, and Selection rules

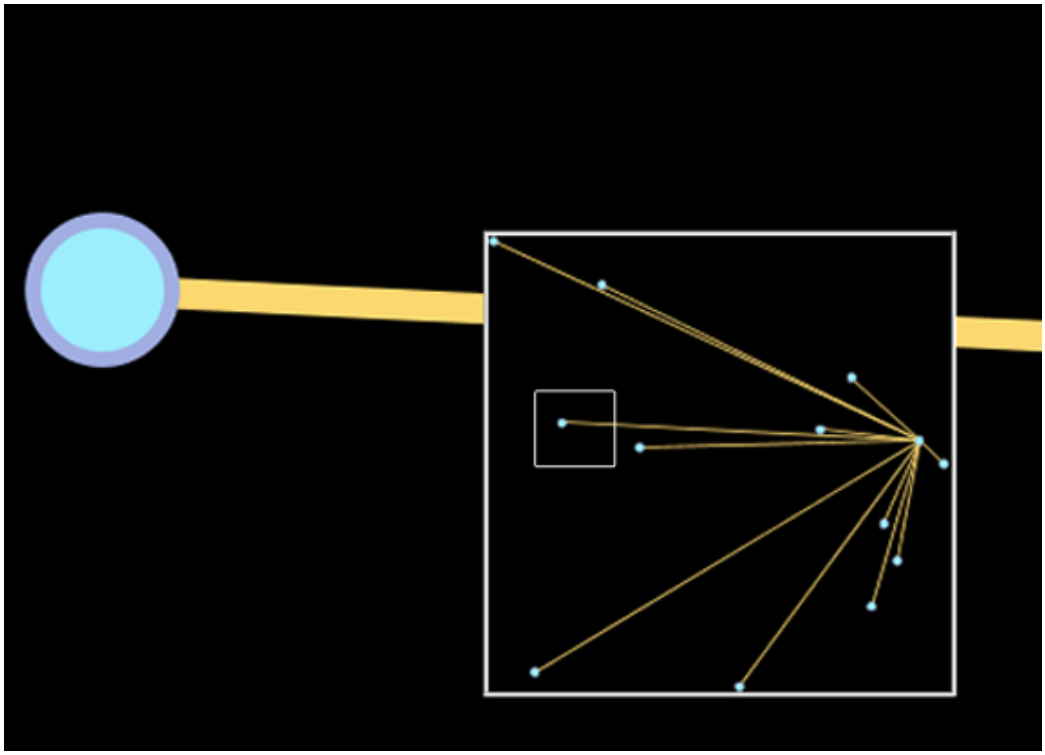

Abbildung 4.2: Übersichtsfenster des Graphen, ebenfalls implementiert wurde hier die Anzeige der aktuellen Position in der Übersicht (innerer weißer Rahmen).

Die andere Möglichkeit um die Zeit zu minimieren ist die Zielfläche zu maximieren. Da eine Bildschirmanzeige aber nur einen begrenzten Platz zur Verfügung stellt, kann man die Zielgebiete (wie Schaltflächen oder Menüs) nicht beliebig groß darstellen. Man kann die Buttons aber am Bildschirmrand positionieren – da der Cursor dort in der Regel festgehalten wird kann man die Zielfläche als unendlich groß betrachten. Die Situation ist an einer neuen Werkzeugleiste, die nun das Menü enthält, in Abbildung 4.3 illustriert.

Das funktioniert allerdings nur, wenn das Programm im Vollbild-Modus ausgeführt wird, da ansonsten die Titelleiste des Fensters[2] den obersten Platz einnimmt. Das Menü wurde so angepasst, dass dieser Speziallfall im Vollbildmodus ermöglicht wird. Um die Geschwindigkeit ansonsten ebenfalls zu verbessern wurden die Menüeinträge auf den gesamten in der Werkzeugleiste vertikal verfügbaren Platz erweitert.

Allgemein sind auch Konsistenz und Einhaltung von Konventionen wichtig für die Effizienz. Das verringert den Lernaufwand, da das Verhalten der

[2]Nicht bei allen Fenster-Managern, siehe z.B. OS X

Abbildung 4.3: Maximierung der Zielfläche nach Fitts durch obere Begrenzung des Cursors.

Oberfläche – u.U. auch aus externen Anwendungen – bereits bekannt ist. Eine wesentliche Konvention, die in Guess fehlt und nun implementiert wurde, ist beispielsweise der Zoom über das Mausrad. Ansonsten sollte man bei Guess nicht auf plattform-spezifische Guidelines wie die „Windows User Experience Interaction Guidelines“, „GNOME 2.0 Human Interface Guidelines“ oder „Apple Human Interface Guidelines“ zurückgreifen, da Guess (durch Java) unter vielen verschiedenen Plattformen läuft und somit je nach Plattform sehr fremd aussehen würde. Durch Umsetzung von großen Teilen der „Java Look and Feel Design Guidelines“ (17) wird nun (mit Kompromissen) die entsprechende Konsistenz auf allen Plattformen gewährt, was zu einer höheren Effizienz beiträgt.

4.1.3 Layout

Betrachtet man das Hauptfenster genauer, so fällt auf, dass relativ viel Platz ungenutzt bleibt. Freier Platz kann gut zur optischen Trennung eingesetzt werden. In Guess wirkt die Oberfläche, auch aufgrund der vielen Rahmen, dadurch unruhig und unübersichtlich.

Der linke und rechte Rand des Hauptfensters sind passende Beispiele: An der linken Seite befindet sich ein 12 Pixel großer Rand ohne jede Funktion. Auf der rechten Seite sind es 2 Pixel. Ein Abschluss des Fensters wird in der Regel schon durch den Fenster-Manager in Form eines Fensterrahmens erzeugt, daher kann der zusätzliche Rand auch wegfallen. An der unteren Seite können Werkzeugleisten wie z.B. die Konsole angedockt werden. Die

Auswahl der entsprechenden Werkzeugleiste geschieht über dort angebrachte Reiter. Oft ist die einzige Werkzeugleiste die Konsole, der einzelne Reiter kann also auch weggelassen werden, falls nur eine Werkzeugleiste angezeigt wird, da keine Auswahl vorgenommen werden kann. Die Konsole selbst ist mehrfach von Rahmen umgeben, hier genügt ebenfalls eine einzige Begrenzung zum Inhaltsbereich. Diese und weitere Abstände und Rahmen wurden entfernt oder vereinfacht.

Am unteren Ende des Fensters befindet sich die Statusleiste (siehe Abbildung 4.4). Diese setzt sich aus folgenden Bestandteilen zusammen:

Abbildung 4.4: Ursprüngliche Statusleiste mit fünf Bereichen welche nachfolgend angesprochen werden.

1. Eine Fortschrittsanzeige, welche während des Programmablaufs (fast) völlig ungenutzt bleibt, jedoch permanent Platz benötigt. Hier wäre ein eigener Dialog zur Fortschrittsanzeige besser geeignet. Der vorhandene Dialog zur Fortschrittsanzeige ist jedoch nicht geeignet, da dort kein Fortschritt angezeigt wird, sondern der Benutzer lediglich alle 30s einen Vorgang abbrechen kann. Im neuen Dialog gibt es die Möglichkeit Vorgänge jederzeit abzubrechen, sowie den wirklichen Fortschritt prozentual anzuzeigen. Ist es nicht möglich den aktuellen Fortschritt zu berechnen, etwa weil der Vorgang unbestimmt lange läuft, so sollte eine andere Darstellung gewählt werden, die lediglich anzeigt, dass der Vorgang in Bearbeitung ist (in Abbildung 4.5 zu sehen). Zusätzlich wäre eine Funktion nützlich, den Graphen während des Vorgangs nicht zu aktualisieren, so könnten aufwändige Vorgänge beschleunigt werden.

2. Eine Liste zur Auswahl eines Zustands, welchen der Benutzer aktiv auswählt. Eine Statusleiste sollte nur Informationen anzeigen und keine Aktionen ermöglichen (18). Zwar wird in der Liste auch eine Information, nämlich der aktuell gewählte Zustand angezeigt, die Hauptaufgabe liegt aber in der Auswahl eines Zustands. Die Liste zum Auswählen des Zustands wurde in eine neue Werkzeugleiste im oberen Bereich des Fensters verschoben. Dabei wurde auch die Funktionalität erweitert, so dass man nun neue Zustände in der Liste speichern kann und zwischen Zuständen navigieren kann. Einige Begrenzungen wurden ebenfalls aufgehoben: So kann der Name eines Zustandes nun aus beliebigen Zeichen und nicht nur aus Buchstaben und Ziffern bestehen. Anstatt nur den Namen eines Zustands anzuzeigen erleichtert ein Bild des Graphen die

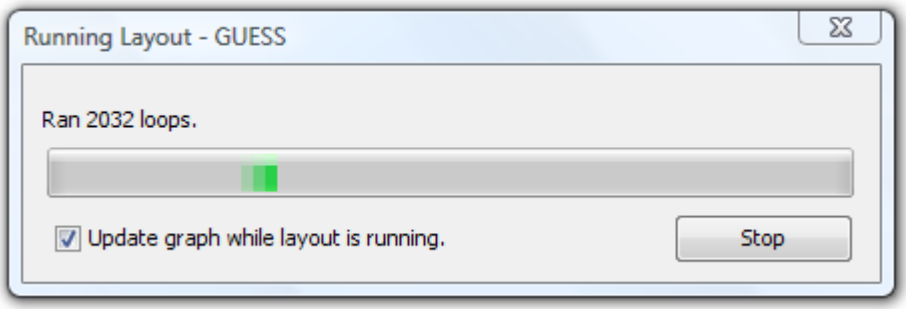

Abbildung 4.5: Neuer Statusdialog wobei hier kein Fortschritt, sondern nur ein aktiver Prozess angezeigt wird, da der Balken nicht gefüllt ist.

Zuordnung. Abbildung 4.6 zeigt dieses neue Element zum Auswahl von Zuständen.

3. Eine Liste von Buttons zur Auswahl eines Modus zum Navigieren oder Bearbeiten. Diese Aktionen sollten ebenfalls in die neue Werkzeugleiste verschoben werden. Es wird hier zwar auch eine Information angezeigt, nämlich das momentan gewählte Werkzeug, hauptsächlich geht es aber wie schon in der Zustandsauswahl um eine Aktion, nämlich die Auswahl eines neuen Werkzeugs. Anstatt permanent fünf Schaltflächen anzuzeigen wird nun nur noch das aktuelle Werkzeug angezeigt. Durch einen Klick auf dieses wird ein Menü geöffnet, welches die Auswahl eines anderen Werkzeugs ermöglicht. Die neue Auswahl ist in Abbildung 4.7 dargestellt.

4. Die Anzeige einer Nachricht, wie etwa eines Hinweises oder eines Fehlers. Diese Anzeige ist die meiste Zeit leer, der Platz bleibt also ungenutzt. Die leere Nachricht wird deshalb ausgeblendet, und nur wenn wirklich eine Nachricht vorliegt, wird diese angezeigt. Je nach Art der Nachricht wird diese nun auch unterschiedlich dargestellt: Eine Fehlermeldung beginnt mit einem Symbol für Fehler und der Text wird in rot dargestellt. Das zeigt dem Benutzer sofort, dass ein Fehler aufgetreten ist. Normale Nachrichten verwenden diese Hervorhebung nicht.

5. Ein Button zur Anzeige einer Übersichtsfensters (siehe Abbildung 4.2). Dieser kann ebenfalls aus den gleichen, wie bereits in den Punkten 2 und 3 aufgeführten, Argumenten in das Menü verschoben werden. In einem vorherigen Abschnitt wurde bereits erwähnt, dass das Übersichtsfenster jetzt auch beim Drücken der mittleren Maustaste angezeigt wird.

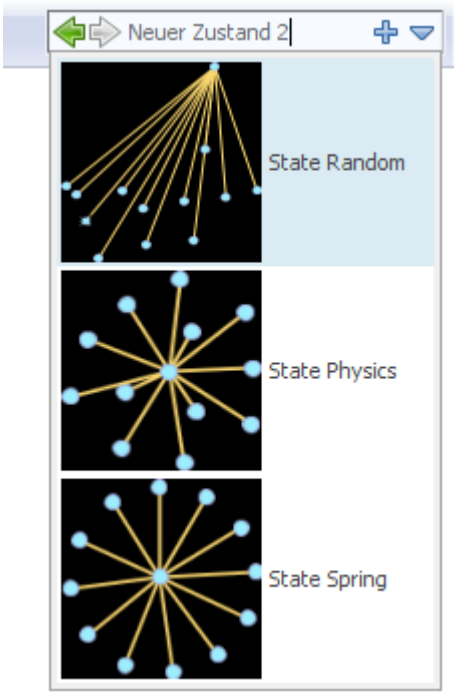

Abbildung 4.6: Neue Zustandsauswahl mit ausgefahrenem Menü.

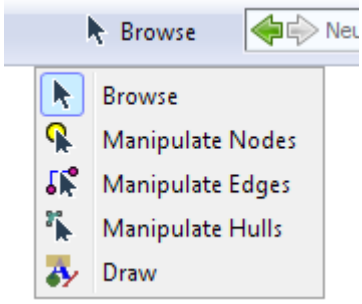

Abbildung 4.7: Neue Werkzeugauswahl mit ausgefahrenem Menü. Momentan ausgewählt ist das „Browse“ Werkzeug zum navigieren im Graph.

Die komplette Statusleiste besteht nun nur noch aus der Nachricht, und diese wird nur dann angezeigt falls eine Nachricht vorliegt. Im oberen Bereich des Fensters wird eine neue Werkzeugleiste erstellt, in welche auch das bisherige Hauptmenü aufgenommen wird. Die neue Werkzeugleiste unterstützt, im Gegensatz zum ursprünglichen Menü, eine Kantenglättung zur besseren Lesbarkeit.

In den Abbildungen 4.8 und 4.9 kann man das ursprüngliche und das angepasste Layout miteinander vergleichen.

Neben dem Hauptfenster bieten die anderen Dialoge weiteres Potential zur Optimierung des Layouts. Die Dialoge beim Start von Guess wurden z.B. in einen neuen Dialog zusammengefasst. Die besprochenen Optimierungen zur Effizienz wurden dort ebenfalls umgesetzt. Der neue Dialog ist in Abbildung 4.10 zu sehen.

Ebenfalls im neuen Startdialog zu erkennen ist ein Programmsymbol und eine dezente Grafik im Kopfbereich. Dadurch wird die Identität des Programms gesteigert. Eine ähnliche Grafik wird im neuen About-Dialog verwendet, ansonsten wurde bewusst auf weitere grafische Elemente verzichtet um den Benutzer nicht abzulenken.

Ein Vergleich zweier Dialoge ist in Abbildung 4.11 zu sehen. Dort sind der ursprüngliche „Field Editor“ und der überarbeitete Dialog abgebildet. Die Liste zur Auswahl des Felds im neuen Dialog kann schneller bedient werden als die Baumstruktur im ursprünglichen Dialog. Ebenso macht die Umbenennung der Schließen-Schaltfläche von „Done“ auf „Close“ klarer, dass beim Drücken der Schaltfläche vorher keine Änderung übernommen wird. Unnötige Elemente des Fensters wie die Ellipsen in Beschreibungen, das Standard-Java-Symbol oder die Minimieren/Maximieren Schaltflächen wurden entfernt.

Bei diesem und allen weiteren Dialogen wurde auf Konsistenz bei den Abständen und Größen geachtet. Alle Felder erhalten passende Beschriftungen und sind über die Tastatur zu erreichen (Mnemoniks). Die Fenstertitel wurden wie in (17) empfohlen in das Format „Aktion - Programmname“ geändert, wodurch sichergestellt wird, dass bei mehreren offenen Fenstern das Richtige in der Taskleiste gefunden werden kann. Des Weiteren sind die Dialoge nun nur in der Größe änderbar (wie z.B. der Dialog „Error Log“) wenn die Steuerelemente des Dialogs sich ebenfalls an eine neue Größe anpassen können. Die JIDE und Jgoodies Bibliotheken wurden auch auf den aktuellsten Stand gebracht. Dadurch wurden Fehler bei der Darstellung der Steuerelemente unter aktuellen Betriebssystemen behoben.

4.2 Visualisierung und Präsentation

4.2.1 Darstellung

Die Darstellung des Graphen besteht aus einer Hintergrundfarbe (standardmäßig schwarz), aus Objekten für die Knoten (meist Kreise), und aus Linien, um die Knoten zu verbinden. Die Farbe der Kanten, Knoten und die Linienfarbe des Knoten kann frei gewählt werden. Ebenso kann eine beliebige

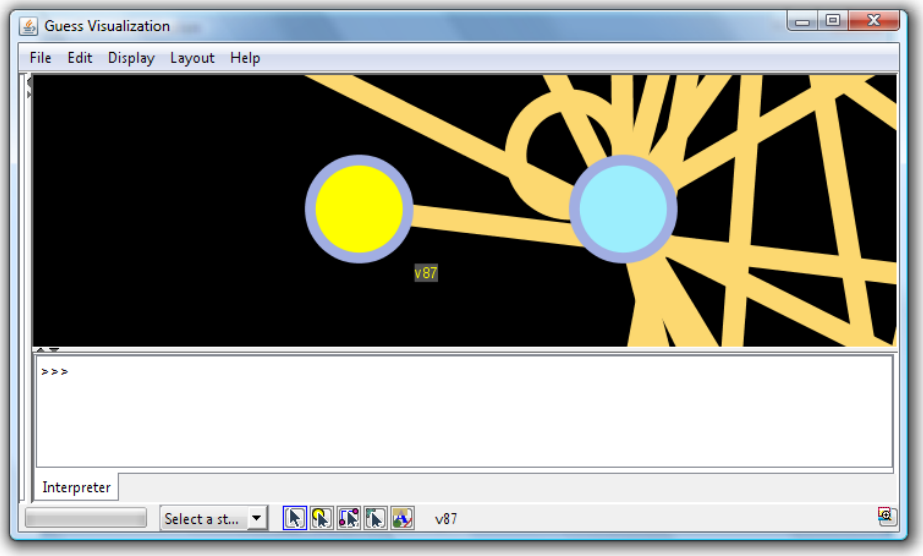

Abbildung 4.8: Ursprüngliches Layout des Hauptfensters.

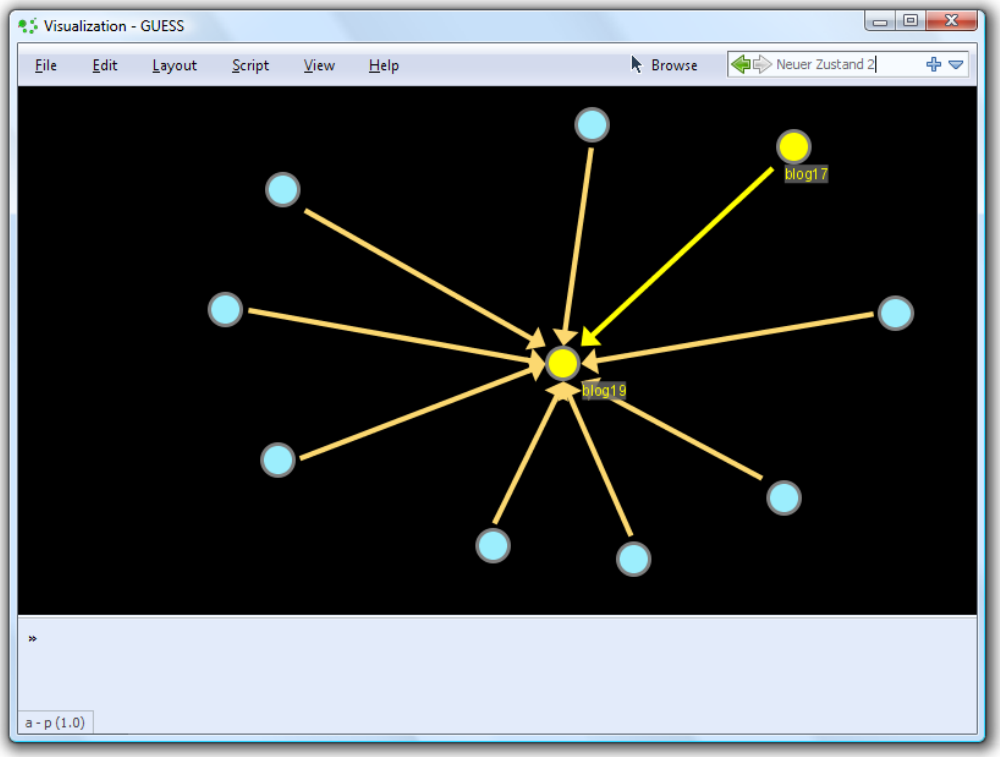

Abbildung 4.9: Neues Layout des Hauptfensters nach den Anpassungen.

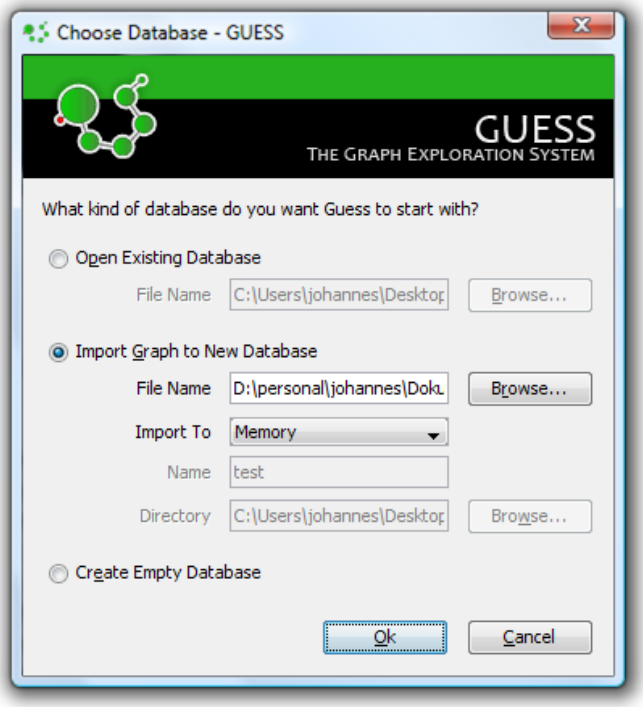

Abbildung 4.10: Neues Layout des Startdialogs.

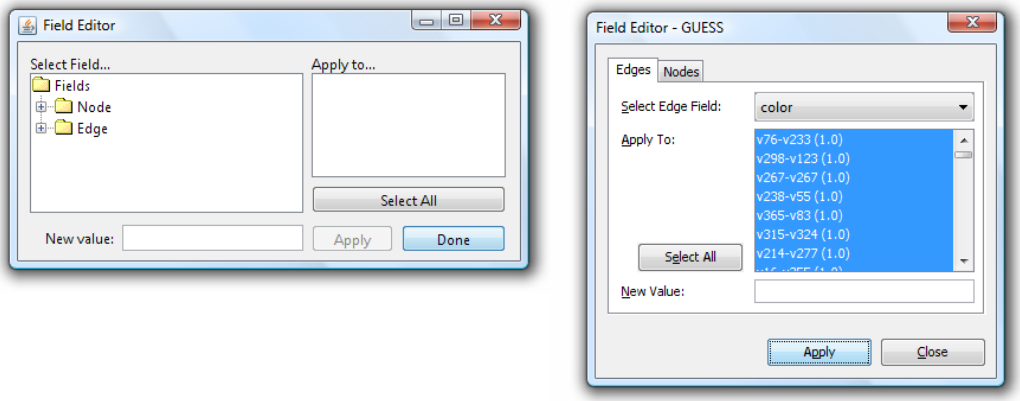

Abbildung 4.11: Vergleich des alten (links) und neuen „Field Editors“ (rechts).

Darstellung aus Abbildung 4.12 für jeden Knoten gewählt werden.

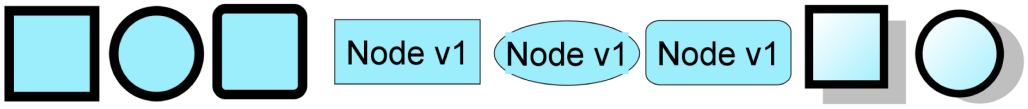

Abbildung 4.12: Standard-Knotenstile, zusätzlich kann auch ein eigenes Bild als Knotenstil verwendet werden.

Da die Farbe der Umrandung der Knoten gewählt werden kann, wurde auch die Möglichkeit hinzugefügt die Linienbreite der Umrandung festlegen zu können.

Bei der Bewegung des Graphen oder der Navigation fällt auf, dass die Linenbreite dicker zu werden scheint. Dieser Effekt tritt auf, da die Kantenglättung bei Bewegung deaktiviert wurde. Das ist sinnvoll um eine höhere Leistung zu erreichen, in der Regel kann man jedoch auf die Abschaltung der Kantenglättung verzichten um eine höhere Darstellungsqualtität zu gewinnen. Um Guess mit großen, rechenintensiven Graphen oder auf leistungsschwacher Hardware nicht zu überfordern, wurde eine neue Möglichkeit implementiert bei Bedarf die Kantenglättung ganz, gar nicht, oder nur für Bewegung zu deaktivieren. Dazu kann beim Start des Programms die gewünschte Qualitätsstufe mit dem Kommandozeilenparameter `--quality` festgelegt werden.

Hält der Benutzer den Cursor über einen Knoten oder eine Kante, so werden Beschriftungen an dem bzw. den jeweiligen Knoten angezeigt. Die entsprechenden Objekte werden farblich hervorgehoben. Dieser Effekt ist auch in den Abbildungen 4.8 und 4.9 zu sehen. Befindet sich ein Knoten zu nah am Rand des Fensters, so wird die Beschriftung abgeschnitten. In diesem Fall sollte die Beschriftung aber besser mehr zur Mitte des Fensters hin verschoben werden. Auch wird jeweils nur der Name des Knoten angezeigt. Es interessieren aber evtl. auch weitere Felder der Knoten, so dass hier eine neue Funktion hinzugefügt wurde um weitere Felder in die Beschriftung aufzunehmen.

Um weitere Möglichkeiten zu erhalten, bestimmte Knoten hervorzuheben oder Sachverhalte eingängig darzustellen, wurde die Darstellung um ein Effektsystem erweitert. Animationen eignen sich hervorragend dazu die Aufmerksamkeit des Benutzers – oder bei einer Präsentation die des Publikums – auf einen Sachverhalt zu lenken. Die Ursache liegt darin begründet, dass der Mensch Bewegungen bevorzugt wahrnimmt (20; 21).

Bei der Implementierung wurde, worauf später noch im Detail eingegangen wird, eine flexible Struktur gewählt um ein leichtes Hinzufügen von wei-

teren Effekten zu ermöglichen. Zur Demonstration wurden drei Effekte entwickelt:

1. Der **Pulse-Effekt** stellt den Knoten als pulsierend, ähnlich einem Herz, dar. Er ist zu vier Zeitpunkten in Abbildung 4.13 abgebildet.

2. Der **Ring-Effekt** kann in zwei Richtungen ausgelöst werden. In beiden Fällen wird jedoch ein Kreis gezeichnet, in dessen Mittelpunkt ein Knoten liegt. Entweder verringert sich der Radius und der Kreis verkleinert sich in Richtung des Knotens oder der Radius vergrößert sich und bewegt sich so vom Kreis fort. Hat der Radius eine entsprechende Größe, so wird der Kreis langsam ausgeblendet. Dadurch kann ein Knoten sehr schnell gefunden werden.

3. Beim **Arrows-Effekt** werden kleine Pfeile auf den Kanten gezeichnet. Die Pfeile bewegen sich ständig in Richtung der gerichteten Kante. Damit kann auf einen Blick auch bei vielen übereinanderliegenden Kanten die Richtung der entsprechenden Kante gefunden werden. Der Effekt ist ebenfalls in Abbildung 4.13 aufgeführt.

Aufgerufen werden können die Effekte über die Konsole per Gython oder direkt über das Kontextmenü der Knoten bzw. Kanten. Ein neues, nachladbares Skript ersetzt bei Bedarf das Standard-Verhalten des Hervorhebens eines Knoten beim Überfahren mit dem Cursor: Die ein- und ausgehenden Kanten werden mit dem Arrows-Effekt ausgestattet und die entsprechenden Knoten werden eingefärbt und mit dem Pulse-Effekt versehen. Die Farbe der Knoten ist bei ein- und ausgehenden Nachbarknoten verschieden.

4.2.2 Übersichtlichkeit

Gerade bei einem Graph mit vielen Knoten kann schnell die Übersichtlichkeit verloren gehen. Durch geeignete Layouts kann man die Überschneidungen von Kanten verringern, aber nicht völlig vermeiden. Befindet man sich noch in einer hohen Zoom-Stufe, so nimmt die Breite der Kanten zu und die Übersicht wird wie in Abbildung 4.14 zu sehen noch weiter beeinträchtigt. Es sind aufgrund von Überlagerung weder alle Kanten einzeln zu identifizieren noch den Knoten eindeutig zuzuordnen.

Um bei Situationen wie in Abbildung 4.14 dargestellt eine bessere Übersicht zu erhalten wird ein neues Zoom-Verfahren eingeführt, welches folgend als „Spacing“ bezeichnet wird. Spacing deshalb, da nur noch die Abstände verändert werden, die Größe der Knoten und die Linienbreite bleibt konstant. In Abbildung 4.15 ist der gleiche Ausschnitt wie in Abbildung 4.14 zu

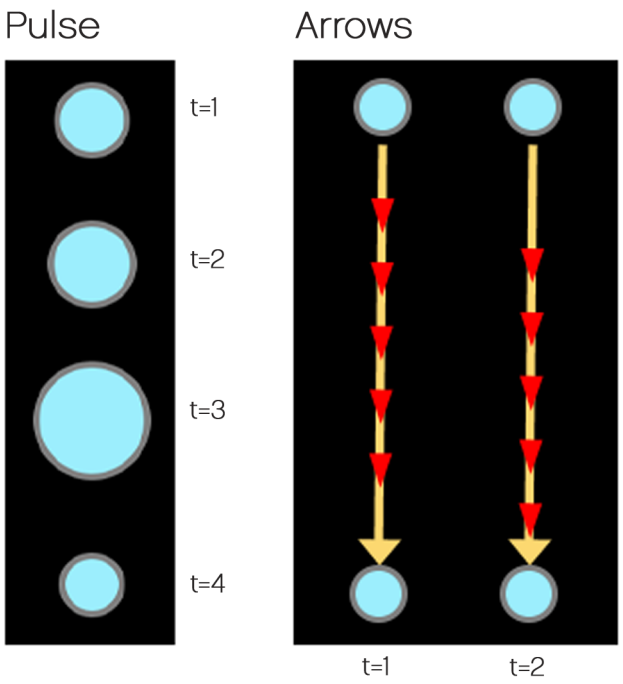

Abbildung 4.13: Die Effekte „Pulse“ und „Arrows“ zu jeweils unterschiedlichen Zeitpunkten t.

sehen. Aufgrund der unverändert großen Knoten und Kanten kommt es aber zu deutlich geringeren Überlagerungsflächen.

Der alte Zoom bleibt ebenfalls erhalten und der Benutzer hat während des Programmablaufs die Möglichkeit je nach Situation zwischen beiden Zoom-Verfahren zu wechseln. Durch Spacing wird jedoch nicht die Komplexität des Graphen an sich verringert. Die Anzahl der Knoten und Kanten bleibt gleich.

Interessiert sich der Benutzer nur für einen Ausschnitt eines Graphen bzw. einen Knoten und dessen Nachbarschaft, so werden trotzdem viele Kanten und Knoten angezeigt, die nichts mit dem gesuchten Ausschnitt zu tun haben. Durch ein neues Layout das folgend als „Neighbourhood-Layout“ bezeichnet wird, kann man genau das erreichen: Das Layout erwartet einen Startknoten und von diesem ausgehend wird nur die direkte Nachbarschaft (Level 1) angezeigt. Um den Ausschnitt einordnen zu können kann auf Wunsch die Nachbarschaft der Nachbarschaft (Level 2) mit einer geringeren Opazität angezeigt werden. Das Layout erwartet also einen Parameter für den gewünschten Level. Eine Anzeige der Nachbarschaft mit einem Level höher als 3 ist so zwar möglich, aber nicht besonders sinnvoll, da die Übersichtlichkeit meist

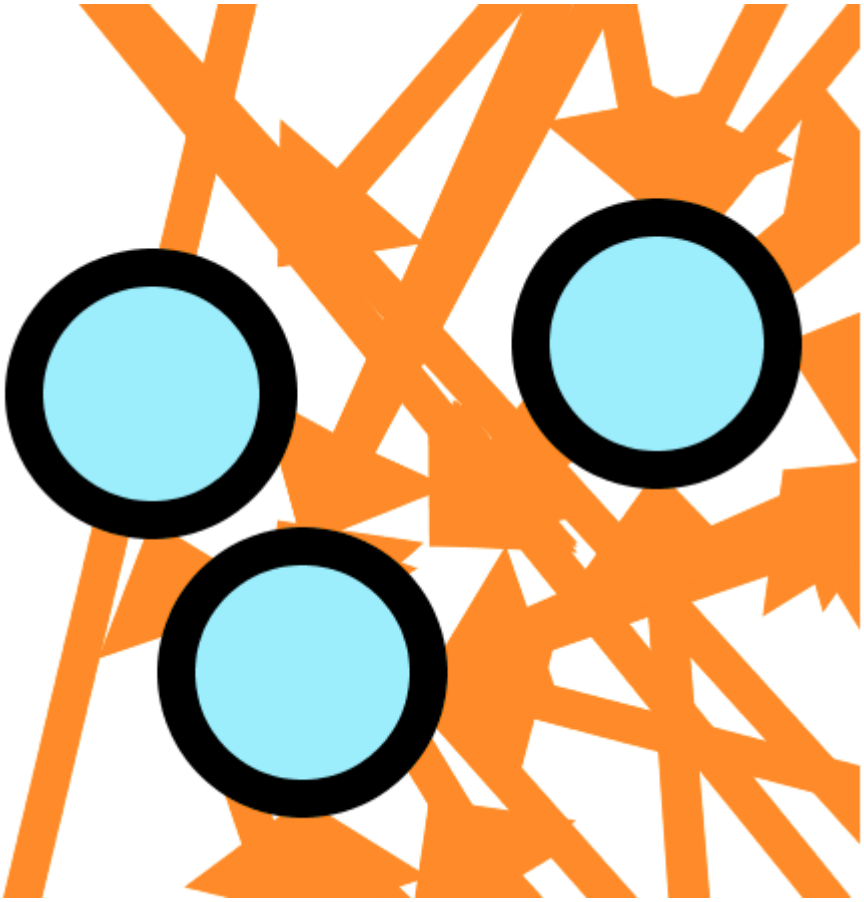

Abbildung 4.14: Graph mit Standard-Zoom in erhöhter Zoomstufe.

nicht besser als die der Ausgangssitutation ist. In Abbildung 4.16 wurde das Neighbourhood-Layout Level 2 auf einen Graph angewendet.

Um das Layout schnell aufrufen zu können wurde es ebenfalls, wie schon das Radial-Layout, in das Kontextmenü der Knoten integriert. Die Layouts werden aus diesem Zweck in netz- und knotenspezifische Layouts unterteilt. Knotenspezifisch bedeutet dabei, dass das Layout ausgehend von einem bestimmten Knoten erzeugt wird, was bei netzspezifischen Layouts nicht der Fall ist. Knotenspezifische Layouts werden zusätzlich zum Hauptmenü im Kontextmenü der Knoten angezeigt.

4.2.3 Präsentation

Zur Präsentation der Ergebnisse ist es wünschenswert eine Funktion zum Export der Graphrepräsentation zu bieten. Es existiert bereits die Möglichkeit

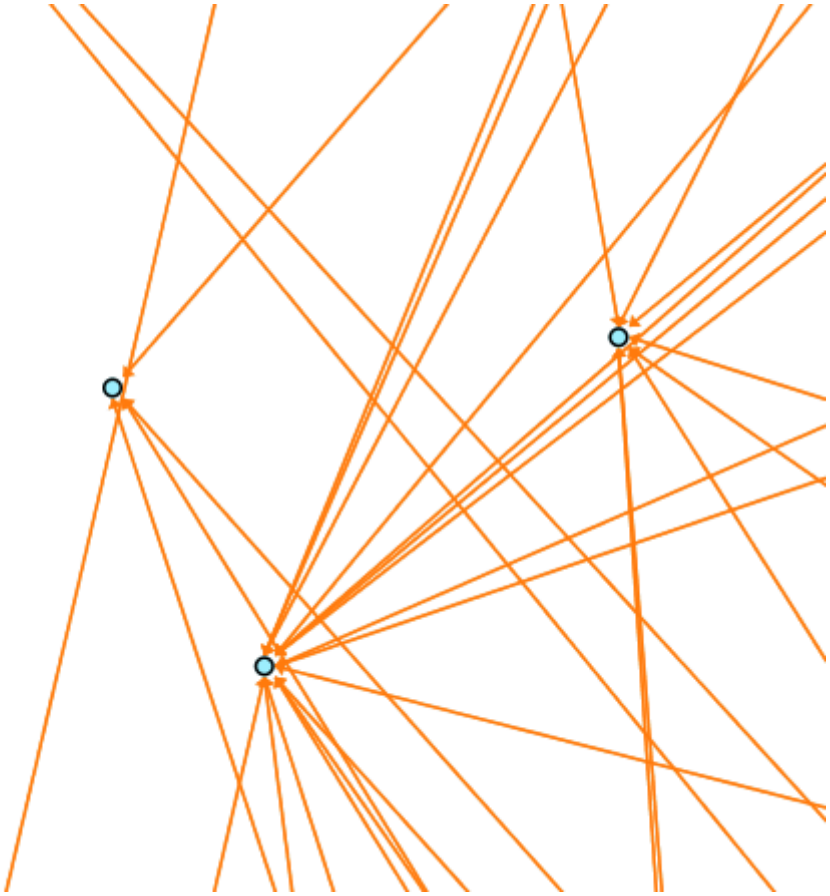

Abbildung 4.15: Spacing bei gleichem Ausschnitt des Graphen und gleicher Zoomstufe wie in Abbildung 4.14.

den Graph als Bild und auch als Video zu exportieren. Der Video-Export beschränkt sich allerdings auf eine Folge von Konsolenbefehlen. Um dies zu erleichtern, wurde eine Werkzeugleiste zur Videoerstellung implementiert (siehe Abbildung 4.17). Dabei muss der Benutzer lediglich eine Folge von Schlüsselbildern erzeugen und gibt dann eine Zeitspanne zwischen je zwei aufeinander folgenden Schlüsselbildern an. Dadurch wird die Erstellung einer Präsentation stark vereinfacht. Zwischen den Schlüsselbildern werden mehrere Zwischenbilder neu berechnet indem das eine Schlüsselbild in das nächste übergeht.

Um Titel für Graphen anzuzeigen wurde die Möglichkeit implementiert in die linke obere Ecke des Inhaltsbereichs einen Text hinzuzufügen. Dieser kann auf Wunsch nach einer bestimmten Zeitspanne langsam ausgeblendet werden. Einer Präsentation können so Überschriften der einzelnen Abschnitte

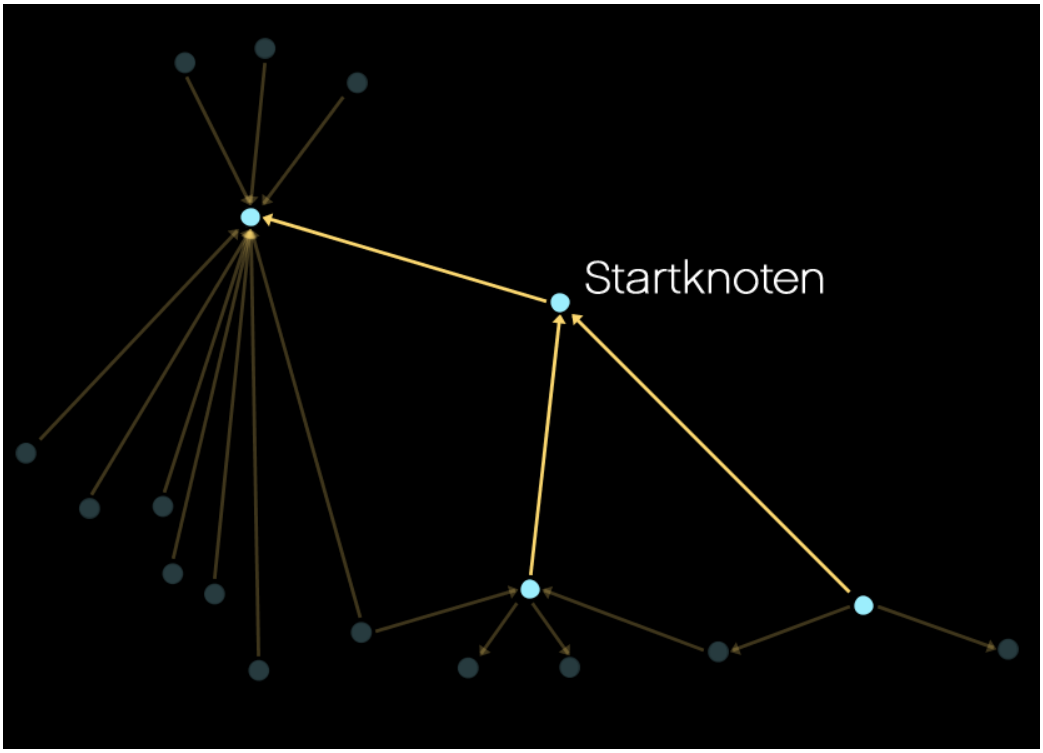

Abbildung 4.16: Neighbourhood Layout Level 2, Startknoten siehe Beschriftung.

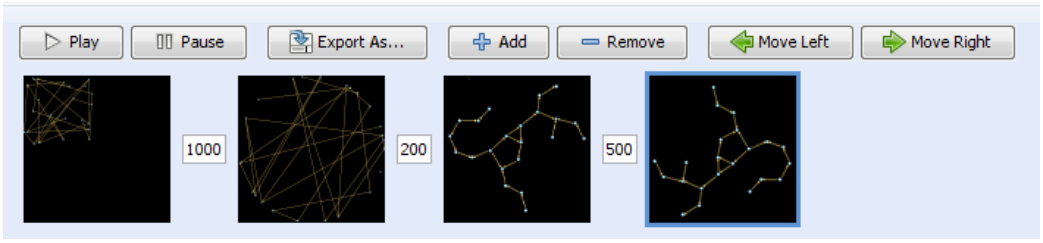

Abbildung 4.17: Werkzeugleiste zu Erstellung von Videos.

zugeordnet werden.

Die im vorherigen Abschnitt zur Darstellung gemachten Änderungen für Effekte und Animation sind natürlich auch für eine Präsentation sehr nützlich.

Kapitel 5

Implementierung

Die Implementierung geschah bis auf wenige Ausnahmen (Gython Skripte) in Java. Dabei wurde falls möglich und sinnvoll auf Entwurfsmuster aus (9) zurückgegriffen. Die Änderungen und Erweiterungen wurden als CVS-Patches veröffentlicht. Zusätzlich wurde eine Version veröffentlicht welche alle Patches integriert.

5.1 Benutzerschnittstelle

5.1.1 Dialoge

Da die Implementierung der Dialoge nur auf eine Auflistung aller Fenster und Steuerelemente hinausläuft, soll darauf nicht im Detail eingegangen werden. Alle in Guess verwendeten Dialoge wurden mit der Swing Bibliothek aufgebaut. Grundsätzlich wird dazu zuerst ein Fenstertyp wie JFrame oder JDialog etc. gewählt auf dem ein Layoutmanager platziert wird. Mit diesem Layout-Manager werden dann die Steuerelemente[1] dem Fenster hinzugefügt und angeordnet. Die Auswahl des Fenstertyps und Layoutmanagers ist von der Aufgabe des Fensters abhängig: Zum Beispiel besitzen die Fenster „Error Log“ und „Choose Database“ den BorderLayout Layoutmanager. Bei einer Änderung der Fenstergröße werden die Elemente und Abstände ebenfalls angepasst. Das Fenster „Error Log“ besteht aus einem JFrame, das Fenster „Choose Database“ aus einem JDialog. Hauptunterschied ist, dass ein JFrame immer parallel zum restlichen Programm ausgeführt wird und ein JDialog modal ausgeführt werden kann. Das bedeutet, dass der aktuelle Programmablauf angehalten wird bis der Dialog geschlossen wird. Um den Steuerelementen ein Verhalten zuzuordnen wird in Swing das Beobachter-

[1] wie z.B. Schaltfächen oder Textfelder

Muster (Observer Pattern) aus (9) erweitert: Den Steuerelementen werden so genannte Listeners hinzugefügt, welche dann beim Auftreten eines Ereignisses aufgerufen werden. So werden einer Schaltfläche Listeners hinzugefügt, welche aufgerufen werden sobald die Schaltfläche ausgelöst wird. Die Listeners enthalten dann das gewünschte Verhalten für die Schaltfläche. Ein Listener ist hierbei ein Objekt, dass die Schnittstelle „ActionListener“ implementiert. Swing wird in Guess durch die Bibliotheken Jgoodies (10) und JIDE (11) erweitert. Diese enthalten weitere Steuerelemente und Designs für die Steuerelemente.

Damit die Felder in den Dialogen nicht jedes Mal erneut ausgefüllt werden müssen, werden diese gespeichert. Dies wurde über das Java Preferences API implementiert. Vorteil ist, dass Daten sehr einfach und plattformunabhängig gespeichert und geladen werden können. Dazu bietet ein Preferences Objekt einfache Methoden wie `put`, `putBoolean` oder `putInt` etc. an, mit deren Hilfe verschiedene (einfache) Datentypen gespeichert werden können. Wo und wie die Daten gespeichert werden[2] ist dabei für die Implementierung nicht weiter relevant. Mit Methoden wie `get`, `getBoolean` und `getInt` werden die Daten einfach wieder geladen. So können nun im Menü des Hauptfensters die letzten 10 gestarteten Skripte in einem so genannten MRU-Menü[3] gelistet werden. Beim Start eines neuen Skripts wird geprüft ob der Dateiname des Skripts bereits in der MRU-Liste enthalten ist. Falls nicht werden die Elemente in der Liste um eine Position nach unten verschoben und der Dateiname des aktuellen Skripts an die 1. Position gesetzt.

Der Status Dialog wurde als JDialog implementiert. Trotzdem wird nicht mit dem Programmablauf gewartet bis der Status Dialog geschlossen wird, sondern es wird parallel dazu ein Vorgang wie z.B. ein Layout ausgeführt. Das ist möglich, da der Vorgang und die Benutzerschnittstelle mit dem Status Dialog in verschiedenen Threads ausgeführt werden. Die restliche Benutzerschnittstelle ist jedoch blockiert, so dass der Vorgang nicht gestört werden kann. Der Vorgang muss also in einem separaten Thread ausgeführt werden. Innerhalb dessen `run()` Methode können der Status der Fortschrittsanzeige und die Beschriftung angepasst werden. Beim Stoppen des Vorgangs wird nicht auf `Thread.stop()` zurückgegriffen, da dies einen inkonsistenten Zustand hinterlassen kann. Vielmehr wird in der Schleife innerhalb der `run()` Methode des Vorgang-Threads mit der Methode `isCanceled()` des Status Dialogs überprüft, ob die Schleife ein weiteres Mal ausgeführt werden soll. Ebenso wird in jeder Schleife mit `isRedrawGraph()` überprüft ob der Graph neu gezeichnet werden soll oder nicht.

[2]Unter Windows meist in der Registrierung, unter Linux meist im HOME-Verzeichnis.
[3]MRU für engl. Most Recently Used

5.1.2 Rückgängig und Wiederholen

Um eine Aktion rückgängig zu machen kann man auf verschiedene Arten vorgehen:

- Man könnte zu jeder Aktion eine inverse Aktion definieren. Zu der Aktion „Verschiebe Knoten v1 um 100 Pixel in X-Richtung“ wäre die inverse Aktion „Verschiebe Knoten v1 um -100 Pixel in X-Richtung“.

- Eine andere Möglichkeit ist es vor jeder Aktion den Zustand der Objekte zu speichern. Dieser Zustand kann dann ggf. wiederhergestellt werden.

Die Vorteile der inversen Aktionen sind ein geringerer Speicherverbrauch und die Möglichkeit die Befehle nicht linear Rückgängig machen zu können. So kann nach vier gemachten Aktionen auch nur die Zweite Rückgängig gemacht werden. Da Guess jedoch über die Konsole Befehle in der Sprache Gython annimmt wird es schwierig zu jeder Aktion eine Inverse zu finden. Aus diesem Grund wird die Rückgängig-Funktion über Zustände implementiert.

Guess kann bereits den aktuellen Zustand des Graphen speichern. In der aktuellen Implementierung wird dies auch effizient gelöst, da nur die Differenz zu dem Anfangszustand gespeichert wird. Es werden aber nicht alle Daten gespeichert die evtl. rückgängig gemacht werden sollen: Lediglich die Knoten und Kanten und ihre jeweiligen Attribute werden gesichert, andere Daten wie Annotationen oder die Hintergrundfarbe können nicht gespeichert werden. Um diese Daten rückgängig machen zu können darf sich die Implementierung nicht allein auf die Zustände des Graphen stützen. Die dazu verwendeten Klassen sind in Abbildung 5.1 eingezeichnet. Bei der Umsetzung wird auch das Befehls-Entwurfsmuster (Command Pattern) (9) verwendet.

Die abstrakte Klasse `GAction` ist der Vorgänger den alle Aktionen erweitern müssen. Dabei müssen nur die abstrakten Methoden `actionContent()`, `getUndoAction()` und `dispose()` überschrieben werden. Durch Verwendung dieser abstrakten Klasse kann man nicht nur Aktionen über die Speicherung von Zuständen rückgängig machen, so wie es die Klasse `GStateAction` macht, sondern auch eine Implementierung mit z.B. inversen Aktionen wäre möglich. Die hier implementierte Lösung beschränkt sich auf das rückgängig machen von Änderungen am Graph durch Aktionen des Menüs, des Kontextmenüs, über das Information Windows, durch aufgerufene Skripte und durch Manipulation von Knoten und Kanten mit dem Cursor. Durch eine neue, weitere Unterklasse von `GAction` könnten auch weitere Aktionen reversibel gemacht werden, wie z.B. Aktionen der Konsole.

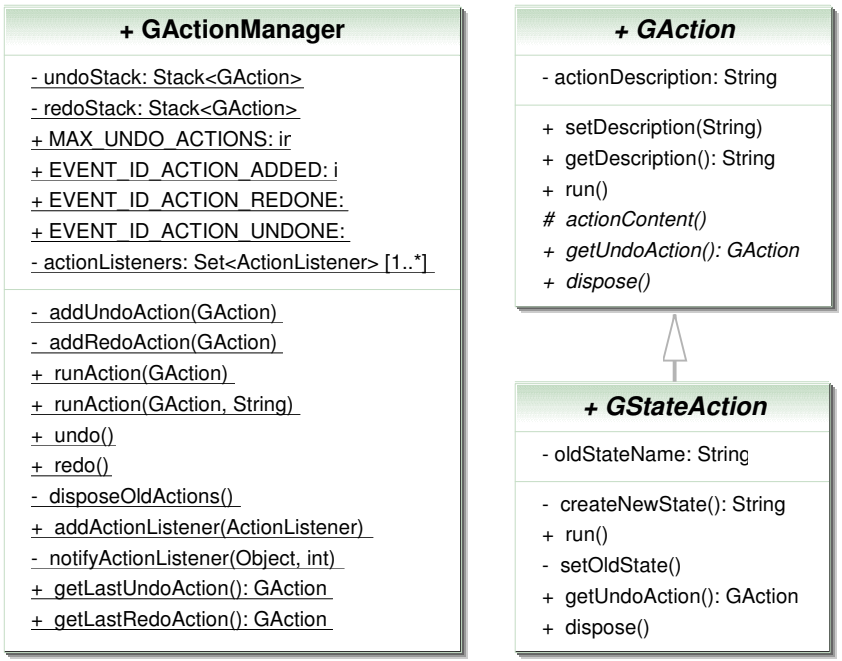

Abbildung 5.1: UML Klassendiagramm um die Implementierung der Rückgängig- und Wiederholen-Funktion darzustellen. Vereinfachte Darstellung.

Die Klasse `GStateAction` ist immer noch abstrakt, da die auszuführende Aktion noch nicht implementiert ist. Für jede Aktion muss eine eigene Klasse definiert werden. Dies kann jedoch auch anonym erfolgen wie in Listing 5.1 zu sehen.

Die statische Klasse `GActionManager` verwaltet die Aktionen, d.h. führt die Aktionen aus, macht die letzte Aktion rückgängig oder wiederholt die zuletzt rückgängig gemachte Aktion. Ebenfalls wird hier das Beobachter-Entwurfsmuster (Observer Pattern) (9) implementiert, da Listener auf Ereignisse wie „Rückgängig ausgeführt“ oder „Wiederholen hinzugefügt“ warten können. Zwei dieser Listeners sind z.B. die Hauptmenüpunkte „Undo“ und „Redo“ im „Edit“ Menü. Falls keine Aktionen zum rückgängig machen vorliegen werden diese Menüelemente deaktiviert. Ansonsten steht die Beschreibung der Aktion in Klammern hinter „Undo“ bzw. „Redo“.

Das Ausführen einer Aktion sieht nun wie in folgendem Beispiel 5.1 zum Anwenden eines Layouts aus:

Listing 5.1: Ausführen der Aktion „Circle Layout“

```
1 GStateAction layoutAct = new GStateAction() {
2         public void actionContent() {
```

```
3                       Guess.getGraph().circleLayout();
4               }
5       };
6       GActionManager.runAction(layoutAct, "Circle Layout");
```

5.1.3 Zustände

Die Funktionalität zum Speichern und Laden eines Zustands war bereits in Guess enthalten. Die Funktionen zum Speichern und Laden sind über das Singleton- und Fabrik-Entwurfsmuster (9) gestaltet, so dass verschiedene Implementierungen denkbar sind. Aktuell wird zum Speichern eine Datenbank genutzt, welche für jeden Zustand eine Tabelle für die Knoten und Eine für die Kanten enthält. Aus diesem Grund darf der Name des Zustands auch nur aus bestimmten Zeichen wie a-Z, 0-9 und _ bestehen. Durch eine Erweiterung lassen sich nun auch beliebige Zeichen wie Leerzeichen oder Umlaute verwenden. Dazu wird das Base32-Verfahren (ähnlich Base64, jedoch nur die Zeichen A-Z und 2-7) verwendet (siehe RFC3548). Bei dieser Codierung werden je drei Byte des ursprünglichen Namens auf 8 Zeichen des neuen Alphabets (A-Z und 2-7) abgebildet.

Ebenfalls wurde intern die Funktion hinzugefügt einen Zustand zu löschen, indem einfach die entsprechenden Tabellen mit den SQL-Befehlen

```
DROP TABLE nodes_<Name des Zustands>
DROP TABLE edges_<Name des Zustands>
```

entfernt werden.

Andere Teile von Guess wollen u.U. auf das Speichern oder Laden von Zuständen reagieren. Dazu wird (unabhängig von der Datenbank Implementierung) die Schnittstelle SStorageEventListener“ definiert. Diese besitzt wie in Abbildung 5.2 zu sehen die Methoden `stateLoaded(String state)` und `stateSaved(String state)`. Nach der Anmeldung des StorageEventListeners z.B. an dem `DBServer` über die Methode `addStorageEventListener` werden die vorhin genannten Methoden beim Laden und Speichern von Zuständen aufgerufen.

Eine konkrete Nutzung ist z.B. beim Wechsel zu einem neuen Zustand dessen Namen als Titel anzuzeigen. Dazu implementiert die Klasse `PFactory` die Schnittstelle `StorageEventListener` wie in Abbildung 5.2 gezeigt. Diese ruft über die Klasse `StorageFactory` die Methode `addStorageEventListener` der Klasse `DBServer` auf, um sich selbst als StorageEventListener zu registrieren. Die Klasse `DBServer` ihrerseits meldet sich beim Speichern oder Laden von Zuständen bei allen registrierten Objekten deren Klassen die

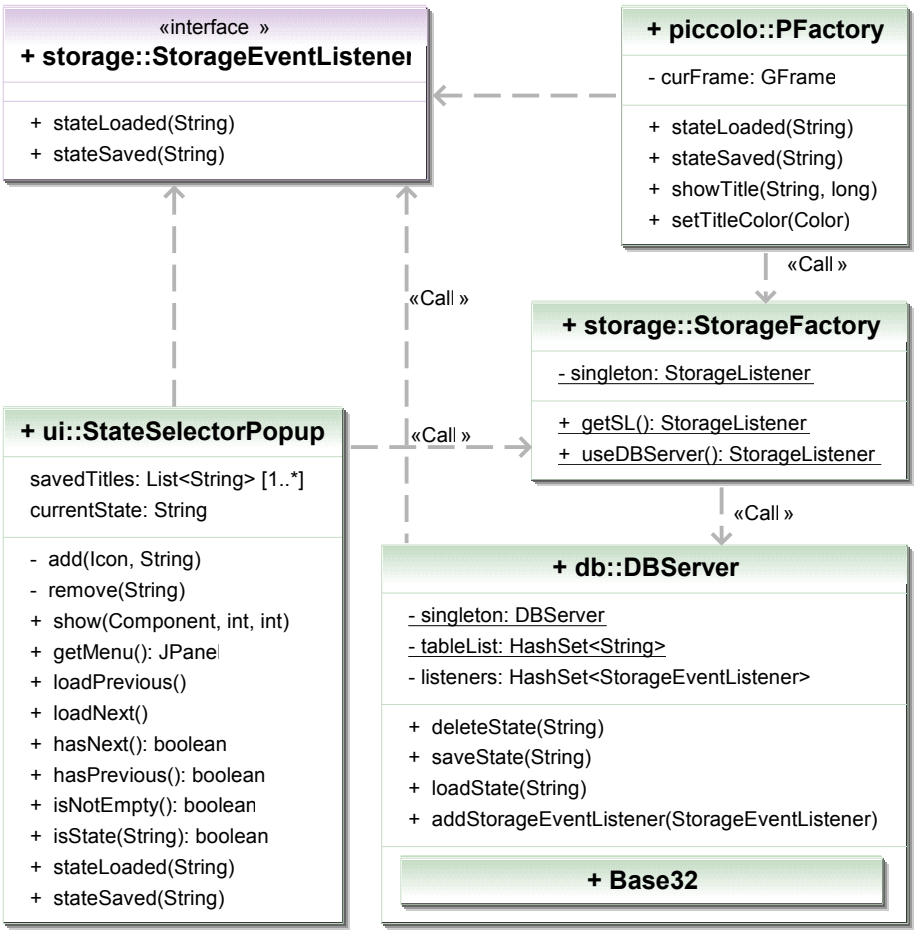

Abbildung 5.2: UML Diagramm der StorageEventListener und der Abhängigkeiten. Vereinfachte Darstellung.

Schnittstelle StorageEventListener implementieren. Dort wird dann die Methode `stateLoaded` bzw. `stateSaved` aufgerufen. Die Klasse `PFactory` implementiert diese durch den Aufruf der ebenfalls neu hinzugefügten Methode `showTitle(<Zustandname>, <Anzeigedauer>)` wodurch der Titel des aktuellen Zustands angezeigt wird. Diese Methode startet einen neuen Thread, welcher den Titel anzeigt und dann wartet, bis die Zeit zum Anzeigen des Titels abgelaufen ist und der Titel langsam ausgeblendet werden kann.

Eine andere Umsetzung des StorageEventListeners ist die Zustandsauswahl. Dort wird der Name des aktuellen Zustands im Steuerelement angezeigt wie z.B. in Abbildung 4.6 zu sehen. Das Popupmenü wird ebenfalls über diesen Mechanismus informiert wenn ein neuer Zustand angelegt wurde. Daraufhin wird ein Vorschaubild des Graphen erstellt und ein neuer Menüeintrag

angelegt. Aus diesem Grund implementiert die Klasse `StateSelectorPopup` auch die Schnittstelle StorageEventListener und ein ähnlicher Vorgang wie bei `PFactory` läuft ab. Die Zustandsauswahl besitzt neben der Schaltfläche zum Anzeigen des Popupmenüs noch fünf weitere Schaltflächen. Im einzelnen sind dies Funktionen zum Laden des letzten und nächsten Zustandes, sowie zum Hinzufügen, Speichern und Laden eines Zustandes. Schaltfächen nicht verfügbarer Aktionen werden ausgeblendet bzw. deaktiviert. Informationen welche Aktionen verfügbar sind werden über die Methoden `hasNext`, `hasPrevious`, `isNotEmpty` und `isState` zurückgegeben.

5.2 Visualisierung und Präsentation

5.2.1 Neighbourhood Layout

Das Neighbourhood Layout zeigt nur die direkte Nachbarschaft eines Knoten an. Auf Wunsch wird die weitere Nachbarschaft der Nachbarschaft mit verringerter Opazität angezeigt. Durch die rekursive Methode `getNeighbours` wird eine Tabelle `elementLevel` mit Elementen und den zugehörigen Leveln erstellt. Der Level bezeichnet dabei die Nähe zum Ausgangsknoten, wobei z.B. 0 dem Knoten selbst und 1 einem direkten Nachbarn entspricht. In Listing 5.2 wird die Methode vereinfacht in Pseudocode dargestellt.

Listing 5.2: Pseudocode für getNeighbours

```
1  Function getNeighbours(Startknoten, Level) {
2  Für alle Knoten v:
3          Falls v noch nicht untersucht wurde:
4                  elementLevel.put(v, Level);
5                  nextNodes.add(next);
6
7  Für alle Kanten e:
8          Falls e noch nicht untersucht wurde:
9                  Falls einer der Knoten von e der
                      Startknoten ist:
10                         elementLevel.put(e, Level);
11
12 Falls Level soweit wie gewünscht ist:
13         fertig;
14 Für alle (neuen) Knoten v in nextNodes:
15         getNeighbours(v, level + 1);
16 }
```

Nachdem die entsprechenden Knoten und Kanten erkannt wurden müssen diese noch eingefärbt werden. D.h. nicht in der Tabelle `elementLevel` aufgeführte Elemente werden ausgeblendet, Elemente mit dem Level 1 werden normal dargestellt, Elemente mit höheren Leveln nur bei Bedarf und je nach Level weniger opak.

Diese und weitere Methoden wurden dazu in einer Klasse `Neighbour` zusammengefasst. Dabei erweitert diese die abstrakte Klasse `AbstractLayout`. Durch Überschreiben der Methode `isIncremental` kann angegeben werden ob das Layout inkrementell berechnet wird, was bei dieser Implementierung aber nicht der Fall ist.

Normalerweise sorgt die Oberklasse `AbstractLayout` auch dafür, dass die Knoten und Kanten nach dem Anwenden des Layouts mit möglichst wenig Überschneidungen angezeigt werden. Dazu werden die Positionen der Knoten entsprechend angepasst. Bei dem Neighbourhood Layout soll sich die Position der Knoten aber nicht verändern, um den erzeugten Ausschnitt ohne Probleme in den Gesamtgraphen einordnen zu können. Daher müssen die entsprechenden Methoden der Oberklasse überschrieben werden, so dass die ursprünglichen Positionen erhalten bleiben.

Um das Layout aufzurufen werden ein Graph, ein Startknoten und der gewünschte Level benötigt. Die Angabe des Graphen kann dabei vernachlässigt werden, da Guess nur auf einen einzigen Graphen ausgelegt ist. Das Layout erwartet den Parameter jedoch um den Quellcode wiederverwendbar zu halten (z.B. bei Programmen zur Graphanalyse mit mehr als einem Graphen). Bleibt also noch die Angabe des Startknotens und des gewünschten Levels, was über die Konsole sehr leicht mit dem Befehl

```
neighbourLayout(<Startknoten>, <Level>)
```

umgesetzt werden kann. Wird das Layout über das Hauptmenü aufgerufen, so erscheint zuerst ein Dialog welcher den Startknoten abfragt. Um diesen Dialog zu umgehen kann das Layout auch direkt über das Kontextmenü des jeweiligen Knoten aufgerufen werden. Dafür wurde die Klasse `NodeEditorPopup` um die Methode `addLayoutItem` erweitert, so dass sich auf einfache Weise knotenspezifische Layouts dem Kontextmenü hinzufügen lassen. Abbildung 5.3 stellt die Situation noch einmal graphisch dar.

Nach dem Aufrufen des Layouts sind u.U. weite Teile des Graphen ausgeblendet. Wendet der Benutzer nun ein weiteres, anderes Layout an, so werden nur die momentan bereits sichtbaren Knoten und Kanten angezeigt, für die interne Berechnung des Layouts spielen die ausgeblendeten Elemente jedoch weiterhin eine Rolle. Es muss also zwischen verschiedenen Layouttypen unterschieden werden. Aus diesem Grund wurde die Schnittstelle

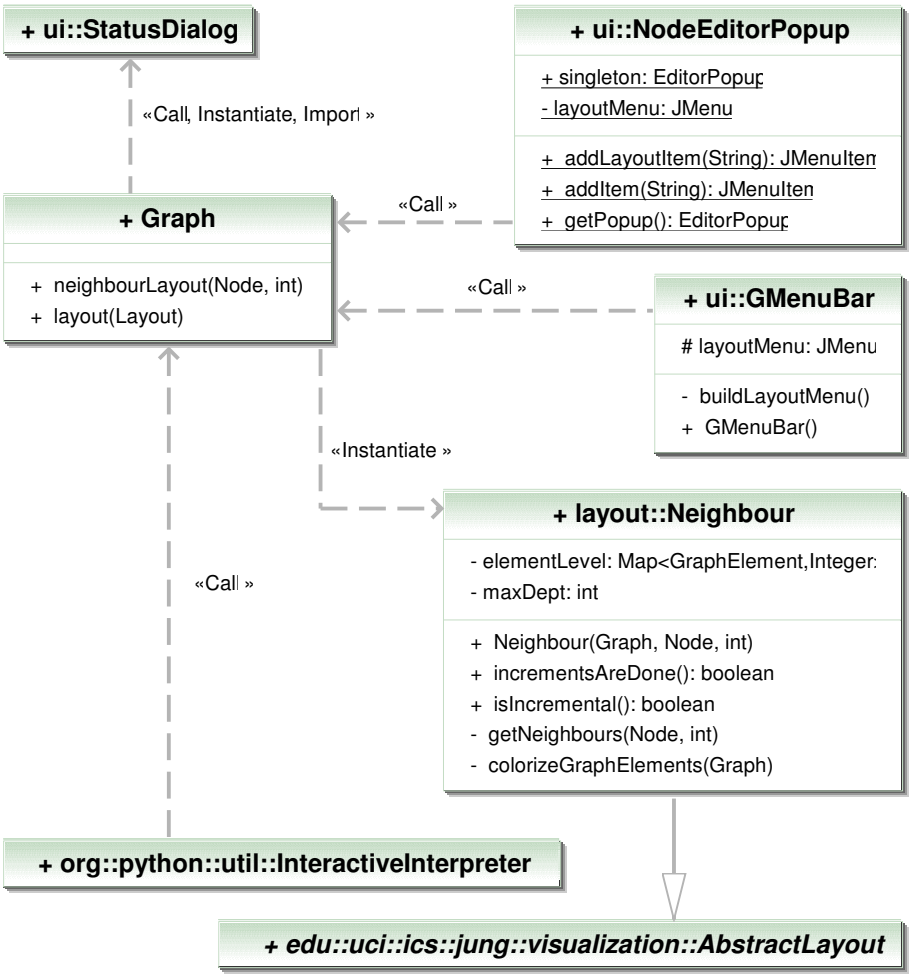

Abbildung 5.3: UML Diagramm um den Zusammenhang des Neighbourhood Layouts darzustellen. Zu sehen ist die Klasse der Konsole (InteractiveInterpreter), des Kontextmenüs (NodeEditorPopup) und Hauptmenüs (GMenuBar) deren Objekte die Methode neighbourLayout des Graphen aufrufen können. Dabei wird ein neues Layoutobjekt instanziiert und der Status Dialog angezeigt. Vereinfachte Darstellung.

`SubGraphLayout` hinzugefügt, die alle Layouts die Teilgraphen anzeigen, implementieren müssen. Der Graph merkt sich nun das zuletzt angewendete Layout. Beim nächsten Anwenden eines Layouts wird die Sichtbarkeit der Knoten und Kanten zurückgesetzt falls das zuletzt ausgeführte Layout die Schnittstelle `SubGraphLayout` implementiert hat.

5.2.2 Spacing

Das Spacing verhält sich wie der normale Zoom, jedoch ändern sich die Knoten nicht in ihrer Größe und die Linienbreite der Kanten bleibt gleich. Beim Spacing ändert sich also scheinbar lediglich der Abstand zwischen den Knoten. Nur scheinbar, da in Wirklichkeit die Koordinaten der Knoten erhalten bleiben müssen. Der Zoom ändert nur den Abstand einer Kamera zu der Bildebene und kann damit sehr effizient in Piccolo durchgeführt werden. Aus diesem Grund wurde das Spacing als angepasste Form des Zooms implementiert, bei dem aber die ungewollten Auswirkungen rückgängig gemacht werden.

Die erste, beim Spacing ungewollte, Eigenschaft des Zooms ist die Veränderung der Linienbreite der Kanten. Dies lässt sich jedoch mit geringem Aufwand beheben: Die Linenbreite wird mit dem Kehrwert des Zoomfaktors (bzw. Skalierungsfaktor) multipliziert. Dabei wird nicht die wirkliche Linienbreite geändert, evtl. möchte der Benutzer diese noch verwenden, sondern lediglich eine nur zur Anzeige verwendete temporäre Variable.

Eine weitere Auswirkung des Zooms ist die Größenänderung der Knoten. Um dies zu verhindern muss etwas mehr Aufwand betrieben werden. Man könnte ebenso wie bei der Linienbreite argumentieren und einfach den Radius mit dem Kehrwert des Skalierungsfaktors multiplizieren. Jedoch hat ein Knoten nicht unbedingt diese Darstellung (siehe auch Abbildung 4.12). Da auch ein eigenes Bild als Knotendarstellung gewählt werden kann muss die Lösung von mehr allgemeiner Natur sein.

Hier bietet es sich deshalb an eine Komposition von affinen Transformationen zu verwenden wie in folgender Aufzählung zu sehen:

1. Eine Translation des Knoten in den Ursprung, wobei der Mittelpunkt des Knotens dem Ursprung entsprechen muss,

2. eine Skalierung des Knoten um den Kehrwehrt des Skalierungsfaktors des Zooms und

3. eine Translation zu den alten Koordinaten, wobei wieder der Mittelpunkt maßgebend ist.

Nun tritt jedoch das Problem auf, dass die Knoten aufgrund der veränderten (Darstellungs-)Größe nicht mehr durch die Kanten verbunden sind. Zur Behebung des Problems müssen die Kanten in ihrer Länge angepasst werden. Jedes Mal alle Kanten neu zu berechnen wäre nicht effizient genug. Die Kantenlängen werden deshalb nur angepasst, falls diese im aktuellen Sichtbereich liegen und sich die Lücke zwischen Knoten und Kante gerade soviel verändert hat, dass der Benutzer es wahrnehmen kann.

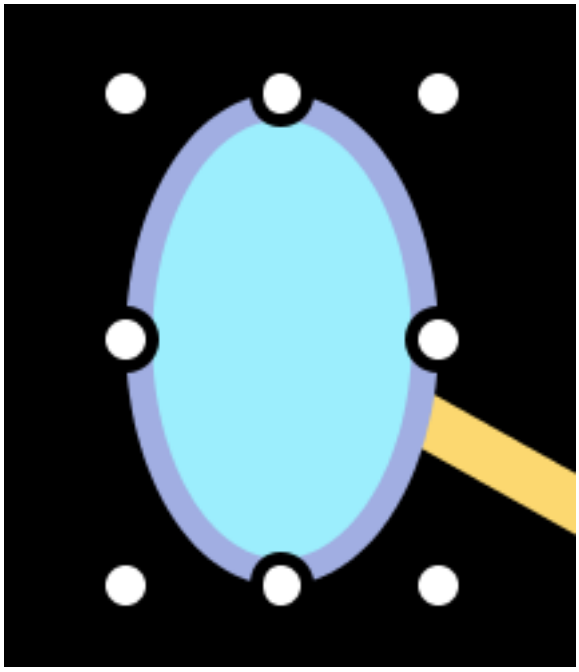

Abbildung 5.4: Die kleinen weißen Kreise werden als Handles (von engl. Griffe) bezeichnet und dienen hier zur Änderung der Größe eines Knoten.

Ein weiteres Problem tritt bei der Manipulation von Knoten und Kanten im Spacing Modus auf: Da die wirkliche Größe und Position nicht mehr mit der Darstellung übereinstimmt werden z.B. die Handles (siehe Abbildung 5.4) an der falschen Stelle gezeichnet. Diese Positionen müssen an die Positionen der Darstellung angepasst werden. Probleme dieser Art gibt es noch weitere (z.B. Abstände und Größe von Pfeilen), die aber jeweils auf die gleiche Problemlösung hinauslaufen und deshalb nicht alle hier aufgeführt werden.

5.2.3 Erweiterte Beschriftungen

Bei den erweiterten Beschriftungen handelt es sich um die Beschriftung, welche beim Überfahren eines Knotens oder einer Kante mit dem Cursor angezeigt wird. Hier wurden zwei Funktionen hinzugefügt: Zum einen wird die Beschriftung nun, sollte der entsprechende Knoten zu nah am Fensterrand positioniert sein, weiter in die Mitte des Bildschirms verschoben. Die dafür nötige Implementierung in der Methode `recalculateLocation` der Klasse `LabelText` verschiebt die Beschriftung gerade um den Wert den diese aus dem Fenster herausragt.

Die zweite neue Funktion ermöglicht ein Hinzufügen von beliebigen Feldern. Möchte man, anstatt nur den Namen des Knoten in der Beschriftung auch die Farbe des Knoten sehen, so kann mit der Methode `addField` das neue Feld hinzugefügt werden. Diese Methode kann nur auf eine einzelne Beschriftung oder auf alle angewendet werden. Dabei wird mit dieser Methode und mit `removeField` eine interne Menge von Feldern `fieldSet` gepflegt. Soll die Beschriftung angezeigt werden, so wird der Text für diese aus den Inhalten der Felder neu zusammengesetzt.

5.2.4 Erzeugung von Videos

Die neuen Funktionen zum Erzeugen von Videos basieren auf Schlüsselbildern. Schlüsselbilder sind dabei Zustände zwischen denen Zwischenbilder berechnet werden können. Diese Berechnung wird in Guess als „morphen“ bezeichnet, genau genommen handelt es sich jedoch nur um ein Tweening (von engl.: in between [dazwischen]) (22; 23).

Zu Beginn muss der Benutzer neue Schlüsselbilder hinzufügen. Dazu wird ein neues Objekt der Klasse `VideoState` erzeugt. Die Methode `createState` übernimmt dabei diese Aufgabe und erzeugt einen Zustand des Graphen und versieht das Objekt mit einem Vorschaubild. Ein VideoState-Objekt hat dabei Attribute für den Namen, das Vorschaubild und die Dauer des Tweenings bis zum nächsten VideoState. Der Name des Zustands hat nur interne Bedeutung, deshalb wird der Name aus einer UUID erzeugt. Der Vorteil der UUID liegt in der für diese Zwecke ausreichenden Eindeutigkeit des Namens.

Bei der Implementierung wird das Model-View-Controller Muster verwendet: Unter Model fallen dabei die VideoState-Objekte, welche in der Liste `videoStates` gespeichert werden. View bezeichnet die graphische Ausgabe der Vorschaubilder und der Schaltfächen für die Benutzerinteraktion, welche über die Methoden `rebuildContent` bzw. `initGUI` erzeugt werden. Den Controller, also die Steuerungsschicht, bilden Funktionen zum Hinzufügen, Verschieben, Löschen, Selektieren, Abspielen, Pausieren und Exportieren.

Beim Abspielen des Videos werden durch die statische Klasse `Morpher` Zwischenbilder erzeugt. Dies geschieht über die Methode `morph` welche jedoch den weiteren Programmablauf blockiert. Da die Methode im gleichen Thread aufgerufen wird in der sich die Benutzerschnittstelle befindet, wird diese somit blockiert. Die Pause-Funktion könnte also nie aufgerufen werden, da der Benutzer keine Möglichkeit hat die Schaltfläche zu bedienen. Aus diesem Grund wurde die Methode `morph` erweitert um eine nicht-blockierende Ausführung zu ermöglichen. Um die Kompatibilität mit alten Aufrufen zu erhalten wird die Methode dabei überladen. Zum Schutz vor Inkonsistenz wird

ein sofortiger Abbruch zwischen zwei Schlüsselbildern nicht erlaubt, d.h. das Morphing wird bis zum nächsten Schlüsselbild fortgesetzt, jedoch ohne währenddessen die Benutzeroberfläche zu blockieren.

Zum Exportieren des Videos wird auf schon bestehende Funktionen zurück gegriffen welche nur geringfügig angepasst wurden (z.B. ein Dateiauswahldialog um einen Namen für die Videodatei anzugeben oder eine Framerate von 25 fps). Die Erzeugung von Videos benötigt relativ viel Speicher, so dass u.U. über einen Kommandozeilenparameter der Java VM[4] mehr Speicher zugeordnet werden muss.

5.2.5 Effektsystem

Durch graphische Effekte lässt sich die Aufmerksamkeit des Benutzers auf einfache Art und Weise auf bestimmte Sachverhalte lenken. Dazu müssen je nach Situation verschiedene Effekte bzw. Animationen zur Verfügung stehen. Es gibt die folgenden Anforderungen an das Effektsystem:

- Die Möglichkeit das Angebot nachträglich um weitere Animationen zu erweitern.
- Eine Implementierung muss dabei die modulare Struktur des Visualisierungssystems berücksichtigen. Eine vollständige Implementierung wurde für Piccolo umgesetzt, soll aber ohne Probleme in anderen Systemen wie z.B. Prefuse umsetzbar sein.
- Mehrere Effekte müssen kombiniert werden können.

Die allgemeinen Anforderungen eines Effekts werden in der abstrakten Klasse `GAnimation` beschrieben: Benötigt werden Funktionen zum Starten und Beenden des Effekts. Um den Code nicht doppelt zu schreiben, sollten ähnliche Effekte zusammengefasst und über Parameter gesteuert werden können. So gibt es beispielsweise eine Ring-Out und eine Ring-In Animation die sich nur in der Richtung des Rings unterscheiden. Über einen Parameter (welcher mit `setAttribute` gesetzt wird) kann dem Effekt hier die Richtung angegeben werden.

Der Effekt soll zum jetzigen Zeitpunkt entweder auf einen Knoten oder auf eine Kante angewendet werden können. Um dies zu ermöglichen wurden die Klassen `GNodeAnimation` und `GEdgeAnimation` abgeleitet. Denkbar ist jedoch auch ein Effekt der sowohl auf Knoten wie auch auf Kanten anwendbar ist, wofür eine neue Klasse von `GAnimation` abgeleitet werden sollte.

[4] Java Virtual Machine

Die eigentlichen Animationensklassen werden nun in jedem Visualisierungssystem von den gerade genannten Klassen abgeleitet. Dort werden für jedes Visualisierungssystem unterschiedliche Grafikbefehle implementiert. In Abbildung 5.5 sind die Klassen `PPulseGAnimation`, `PRingGAnimation` und `PArrowsGAnimation` für Piccolo zu sehen. Um einen weiteren Effekt für Knoten hinzuzufügen, muss lediglich eine Klasse von `GNodeAnimation` abgeleitet, welche die Methoden `start` und `stop` implementiert.

Es gibt nun also für jedes Visualisierungssystem eigene Objekte. Durch den Einsatz des Fabrik-Entwurfsmusters (Factory Pattern) (9) können Objekte aus verschiedenen Visualisierungssystemen erzeugt werden ohne diese selbst kennen zu müssen. Dazu dient die neu hinzugefügte, abstrakte Klasse `AnimationFactory` mit den abstrakten Methoden `generateNodeAnimation` und `generateEdgeAnimation`, welche jeweils den Namen eines Effekts benötigen. Namen werden den Effekten zugeordnet, indem diese zu Beginn bei der `AnimationFactory` registriert werden. Die `PAnimationFactory` implementiert eine solche Fabrik und kann damit Instanzen der bereits genannten Effekte erzeugen. Im Konstruktor der Piccolo Fabrik werden diese Effekte gleich registriert.

Über die Konsole oder das Kontextmenü können die Effekte nun aufgerufen werden. Zu diesem Zweck steht eine Methode `animate` in den Klassen `Edge` und `Node` bereit, welche den Namen des Effekts erwartet. Die Methode ist überladen um zusätzlich eine Zahl zu akzeptieren, welche die Anzahl der Wiederholungen des Effekts begrenzt. Der Aufruf dieser Methode erzeugt über die `AnimationFactory` ein neues Objekt der Klasse `GNodeAnimation` bzw. `GEdgeAnimation` und startet den Effekt. Das Objekt wird dann in die interne Menge `runningAnimations` aufgenommen um den Effekt bei Bedarf stoppen zu können.

Abbildung 5.5 zeigt ein vereinfachtes UML Klassendiagramm um die Situation grafisch zu erläutern.

Die eigentliche Implementierung der Effekte läuft in Piccolo über so genannte „Activities“ ab. Über Activities können bestimmte Animationen interpoliert werden, indem ein Anfangs- und Endzustand eines grafischen Objekts angegeben wird. Beispiele hierfür wären Interpolation zwischen zwei unterschiedlichen Größen eines Objektes, oder verschiedener Opazität. Des Weiteren muss die Dauer und Art des Übergangs angegeben werden. Verfügbare Übergangstypen sind z.B. Linear oder von „langsam zu schnell zu langsam“. Diese Operationen laufen jedoch alle nicht mehr auf den Knoten oder Kanten ab, sondern nur auf den jeweiligen grafischen Repräsentanten.

Um ein sinnvolles Beispiel für die Effekte zu geben wurde ein Skript[5] geschrieben, welches die Standard-Hervorhebung ersetzt. Beim Überfahren eines grafischen Elements mit dem Cursor werden zugehörige Nachbarknoten mit dem Pulse-Effekt versehen, die ein- und ausgehenden Kanten zeigen die Richtung durch den Arrows-Effekt an. Zusätzlich werden die ein- und ausgehenden Knoten unterschiedlich gefärbt. Der Ablauf des Skripts ist relativ simpel: Zuerst wird die Standard Hervorhebung deaktiviert und durch die folgende Neue ersetzt. Beim Eintreffen des Cursors auf dem Element werden die Farben der zugehörigen Elemente gespeichert und geändert, sowie die entsprechenden Effekte gestartet. Verlässt der Cursor das Element wieder werden die Effekte beendet und die Farben wiederhergestellt.

[5]Zu finden unter „newhighlight.py“

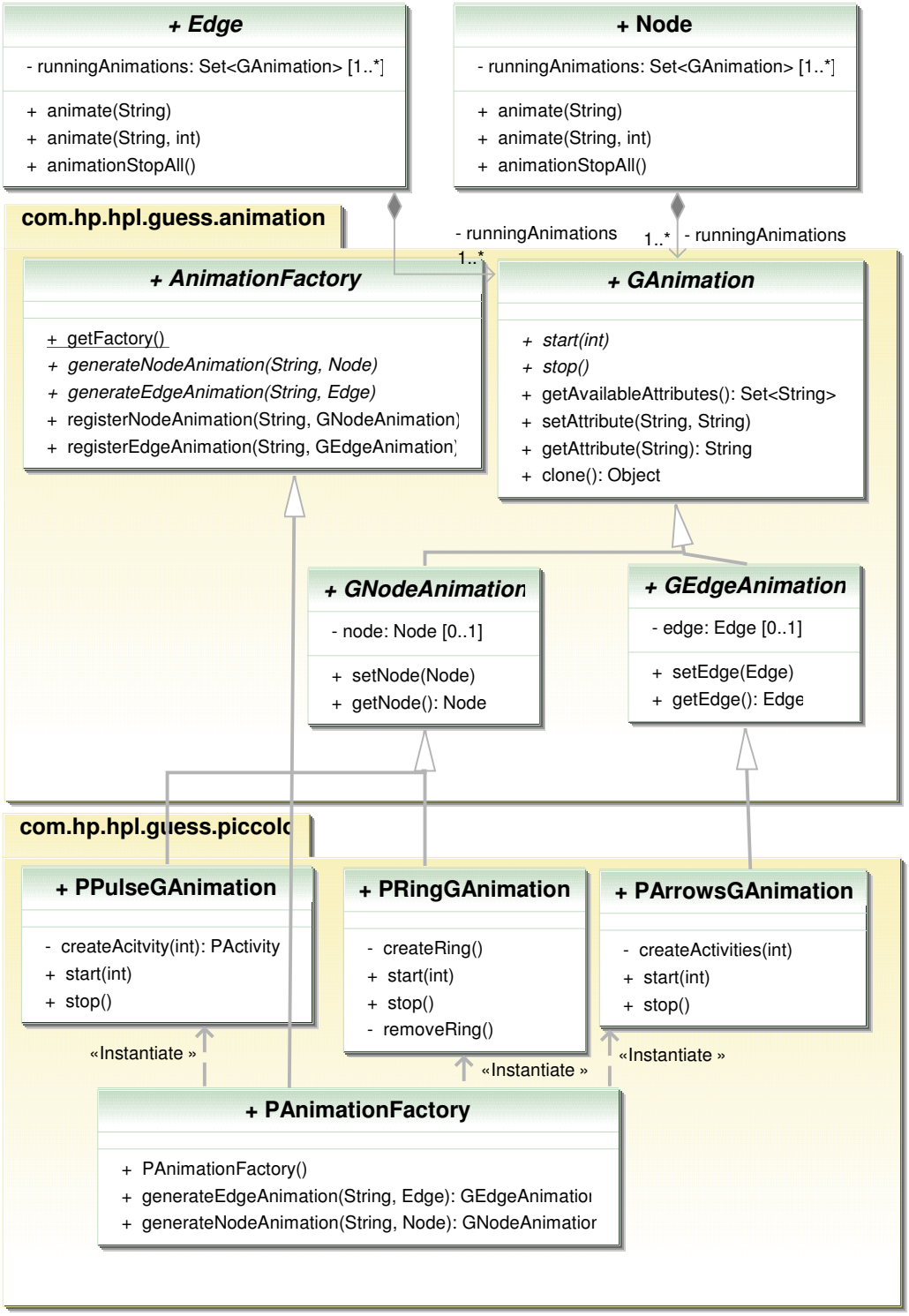

Abbildung 5.5: Vereinfachtes UML Diagramm des Effektsystems.

Kapitel 6

Fazit

6.1 Zusammenfassung

Durch diese Arbeit wurden Veränderungen und Erweiterungen an dem Programm Guess vorgenommen, wodurch der Benutzer bei der Analyse von Graphen besser unterstützt wird. Durch die Teilnahme an im Internet verfügbaren Diskussionsgruppen über dieses Programm und daraus entstandenen E-Mail-Kontakten konnte eine ständige Rückmeldung während der Analyse- und Implementierungsphase stattfinden.

Die Bedienung ist nun einfacher und vor allem die Speicherung bereits gemachter Eingaben wurde von vielen Benutzern als große Erleichterung empfunden. Es gibt nun neue Funktionen um Aktionen auf einfache Art und Weise rückgängig zu machen oder einen Vorgang zu einem beliebigen Zeitpunkt abzubrechen. Die Benutzeroberfläche ist nun konsistent und um Funktionen zur effizienteren Nutzung wie z.B. den Zoom per Mausrad, die Tastenkürzel oder das Skript-MRU-Menü erweitert worden. Mehrere Dialoge wurden zusammengefasst und vereinfacht, so dass diese schneller bearbeitet werden können.

Das Hauptmenü in der neuen Werkzeugleiste wurde restrukturiert und um einige Befehle erweitert. Außerdem wurden Aktionen wie die Zustands- und Werkzeugauswahl aus der Statusleiste überarbeitet und erweitert und in die Werkzeugleiste verschoben. Da die Statusleiste nun nur noch angezeigt wird, falls wirklich eine Nachricht vorliegt, wird hier mehr Platz für den Inhaltsbereich geschaffen. Weiterer, zusätzlicher Platz konnte an anderen Stellen des Hauptfensters gefunden werden.

Auch die Darstellungsqualität des Graphen konnte durch permanente Kantenglättung und besser positionierte und erweiterte Beschriftungen erhöht werden. Die Einbindung von neuen Funktionen wie dem Neighbourhood

Layout und dem Spacing sorgt für eine größere Übersichtlichkeit. Des Weiteren wurden dem Programm neue Funktionen wie das Effektsystem oder die vereinfachten Video-Erzeugung hinzugefügt.

Alle Implementierungen wurden möglichst so gestaltet, dass eine Erweiterung oder Anpassung mit geringem Aufwand verbunden ist.

6.2 Diskussion

Aufgrund des Mangels an Ressourcen können nicht alle Bereiche der Entwicklung vollständig abgedeckt werden. Folgend nun einige Kritikpunkte an der durchgeführten Arbeit:

Dass die neue Benutzeroberfläche der ursprünglichen Fassung wirklich überlegen ist wurde nicht verifiziert, sondern basiert lediglich auf den subjektiven Aussagen einiger Benutzer. Diese Aussage kann erst validiert werden, nachdem Untersuchungen an der Benutzeroberfläche mit mehreren Versuchsgruppen erfolgreich durchgeführt wurden.

Alle gemachten Änderungen wurden auf die Desktop-Version von Guess bezogen. Da sich das Programm aber auch als Applet in einem Browser einsetzen lässt, können einige der Änderungen für die Applet-Version nicht optimal sein.

Zwar wurden alle Änderungen so implementiert, dass eine Anpassung an andere Visualisierungssysteme problemlos möglich sein sollte, getestet werden konnte dies jedoch nicht. Der Grund hierfür ist die Inkompatibilität der ursprünglich verfügbaren CVS Version von Guess mit anderen Visualisierungssystem als Piccolo. Um weitere Systeme wie Prefuse oder Touchgraph verwenden zu können müssen erst Fehler bei deren Einbindung in Guess korrigiert werden.

6.3 Ausblick

Der Quellcode wurde unter einer Open-Source Lizenz im „CVS Repository“ des Projekts (16) veröffentlicht. Damit ist der Quellcode, im Rahmen der GPL Lizenz, öffentlich verfügbar und die Weiterentwicklung der Software steht jedem offen. Neben den Änderungen dieser Arbeit sind natürlich noch weitere Verbesserungen in allen Bereichen von Guess denkbar.

Zunächst könnte man die JUNG Bibliothek durch die aktuelle Version derselben ersetzen, um mit den aktuellen Entwicklungen der Graphen- und Netzwerktheorie Schritt zu halten.

In der Benutzeroberfläche könnte man das „Information Window“ durch

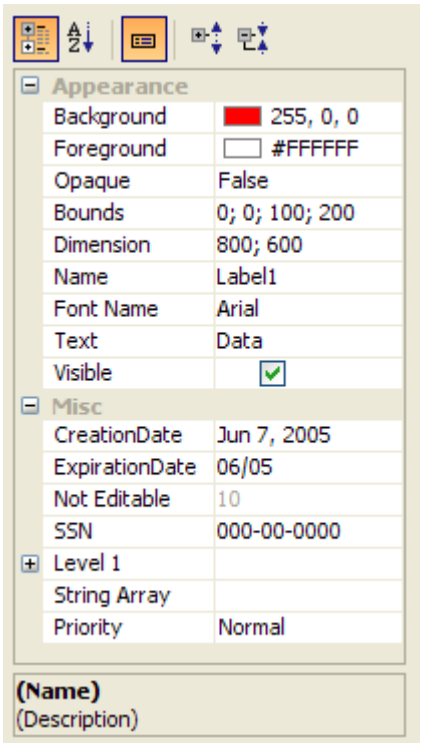

Abbildung 6.1: PropertyPane Komponente aus einer Sammlung der so genannten JIDE Grids (11).

ein „Property Pane“ wie in Abbildung 6.1 zu sehen ersetzen. Dadurch würde die bereits begonnene Minimierung von Maus-Tastatur-Wechseln weiter verbessert. Zum Beispiel müsste beim Ändern eines Wertes von „true“ nach „false“ kein Text mehr eingegeben werden, sondern dieser könnte durch ein Kontrollkästchen ersetzt werden (In Abbildung 6.1 in der Zeile „Visible“ zu sehen). Ebenso müssten Farben nicht mehr als RGB-Wert oder Namen eingegeben werden sondern könnten über ein entsprechendes Popup-Fenster ausgewählt werden.

Die Bedienung der Konsole könnte durch eine automatische Vervollständigung ähnlich zu IntelliSense in Microsoft Visual Studio vereinfacht werden. Dadurch müssten nicht ständig alle Objekte, Befehle, Parameter und deren Reihenfolge gemerkt oder nachgeschlagen werden.

Um die Entstehung eines Layouts besser nachvollziehen zu können wäre es sinnvoll den Übergang zu diesem anzupassen. Momentan wird das alte Layout einfach durch das Neue ersetzt oder bei inkrementellen Layouts ersetzt jeder Schritt den Vorherigen. Ein fließender Übergang, wie bei dem bereits vorhandenen Tweening, würde die explorative Analyse weiter vereinfachen.

Abbildungsverzeichnis

Literaturverzeichnis

[1] E. Adar. Guess: a language and interface for graph exploration. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 791–800, New York, NY, USA, 2006. ACM.

[2] E. Adar and J. R. Tyler. Zoomgraph. *Information Dynamics Lab, HP Laboratories CA–USA*, 2003.

[3] V. Batagelj and A. Mrvar. Pajek - program for large network analysis. *Connections*, 21:47–57, 1998.

[4] E. M. Borgatti, S.P. and L. Freeman. *UCINET 5 for Windows: User's Guide*. Analytic Technologies, Inc., 1999.

[5] E. M. Borgatti, S.P. and L. Freeman. *Ucinet for Windows: Software for Social Network Analysis*. Harvard, MA: Analytic Technologies, 2002.

[6] S. Card, T. Moran, and A. Newell. *The Psychology of Human-computer Interaction*. Lawrence Erlbaum Associates, 1983.

[7] R. Diestel. *Graphentheorie*. Springer Verlag, Heidelberg, 3. edition, 2006.

[8] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software- Practice and Experience*, 21(11):1129–1164, 1991.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, München, 2004.

[10] Jgoodies. :: JGOODIES :: Java User Interface Design. http://jgoodies.de, 2008. [Online; Stand 11. November 2008].

[11] JIDE Software, Inc. JIDE Software - The Best Java and Swing Components Library - Over 100 Components. http://www.jidesoft.com, 2008. [Online; Stand 11. November 2008].

[12] A. Kerren, A. Ebert, and J. Meyer. *Human-centered Visualization Environments GI-Dagstuhl Research Seminar, Dagstuhl Castle, Germany, March 5-8, 2006: Revised Lectures*. Springer-Verlag, 2007.

[13] J. Scott. *Social Network Analysis*. SAGE Publications, London, Thousand Oaks, New Delhi, 2. edition, 2000.

[14] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res*, 13(11):2498–2504, November 2003.

[15] B. Shneiderman and C. Plaisant. *Designing the user interface : strategies for effective human-computer interaction*. Pearson/Addison-Wesley, Boston [u.a.], 4. ed. edition, 2005.

[16] SourceForge, Inc. SourceForge.net: GUESS. http://sourceforge.net/projects/guess, 2008. [Online; Stand 11. November 2008].

[17] Sun Microsystems Inc. *Java Look and Feel Design Guidelines*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2. edition, 2001.

[18] Sun Microsystems Inc. *Java Look and Feel Design Guidelines: Advanced Topics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[19] The Tango Desktop Project. Tango Desktop Project. http://tango.freedesktop.org, 2008. [Online; Stand 11. November 2008].

[20] A. Treisman. Properties, parts, and objects. *Handbook of perception and human performance*, 2:1–70, 1986.

[21] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann, 2004.

[22] Wikipedia. Tweening — Wikipedia, Die freie Enzyklopädie. http://de.wikipedia.org/w/index.php?title=Tweening&oldid=33788380, 2007. [Online; Stand 6. November 2008].

[23] Wikipedia. Morphing — Wikipedia, Die freie Enzyklopädie. http://de.wikipedia.org/w/index.php?title=Morphing&oldid=52272073, 2008. [Online; Stand 6. November 2008].