Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Virginio Carfagno

Evaluation des Google Web Toolkits durch Entwicklung einer ajaxbasierten Mind-Mapping-Anwendung

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Virginio Carfagno

Evaluation des Google Web Toolkits durch Entwicklung einer ajaxbasierten Mind-Mapping-Anwendung

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Softwaretechnik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Kai von Luck
Zweitgutachter : Prof. Dr. rer. nat. Christoph Klauck

Abgegeben am 29. März 2007

Virginio Carfagno

Thema der Diplomarbeit

Evaluation des Google Web Toolkits durch Entwicklung einer ajaxbasierten Mind-Mapping-Anwendung

Stichworte

Google Web Toolkit, Ajax, Benutzerschnittstellen, Entwurfsprinzipien, Vektorgrafiken, Mind-Mapping-Software

Kurzzusammenfassung

Im Rahmen dieser Diplomarbeit wurde durch die Entwicklung einer Mind-Mapping-Anwendung mit dem Google Web Toolkit (GWT) gezeigt, dass die Entwicklung grafisch mittelmäßig anspruchsvoller 2D-Ajax-Anwendungen mit dem GWT im Speziellen und Ajax im Allgemeinen heute noch nicht möglich ist, weil die heutigen populären Web Browser Firefox und Internet Explorer noch keine gemeinsame Vektorgrafiktechnologie nativ unterstützen. Die entwickelte Mind-Mapping-Anwendung läuft nur im Firefox. Des Weiteren wurde durch die Entwicklung der Mind-Mapping-Anwendung gezeigt, dass GWT-Anwendungen nach State-of-the-Art-Entwurfsprinzipien, wie z.B. Modularisierung, Architekturmuster und Entwurfsmuster gebaut werden können.

Virginio Carfagno

Title of the paper

Evaluation of the Google Web Toolkit by the Development of an Ajax based Mind Mapping Application

Keywords

Google Web Toolkit, Ajax, user interfaces, design principles, vector graphics, mind mapping software

Abstract

In the context of this thesis, a Mind Mapping application has been developed by using the Google Web Toolkit (GWT). This has shown that the development of graphically middle-rate sophisticated 2d Ajax applications is not yet possible by using the GWT in particular and Ajax in general, since the currently popular web browsers Firefox and Internet Explorer do not support a common technology for vector graphics natively. The developed Mind Mapping application runs on Firefox only. Furthermore the development of the Mind Mapping application has shown that GWT applications can be built by state of the art design principles such as modularisation, architectural patterns and design patterns.

Danksagung

Ich möchte mich bei allen bedanken, die mich bei dieser Arbeit unterstützt haben.

Besonderer Dank gilt Herrn Professor Dr. rer. nat. Kai von Luck für die Betreuung dieser Diplomarbeit und Herrn Professor Dr. rer. nat. Christoph Klauck seitens der Hochschule für Angewandte Wissenschaften Hamburg.

Der größte Dank gilt meiner Familie für die moralische Unterstützung und insbesondere meinem Bruder Fabio für das Korrekturlesen.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

Seit einigen Jahren ist im Web ein spürbarer Wandel bemerkbar. Im Februar 2004 wurde während eines Konferenz-Brainstormings zwischen O'Reilly und MediaLive für diesen Wandel der Begriff Web 2.0 eingeführt [O'Reilly (2005)].

Unter den Begriff Web 2.0-Anwendungen fallen z.B. Rich Internet Applications (RIA) [O'Reilly (2005)]. RIAs sind Webanwendungen, die große Ähnlichkeit zu Desktopanwendungen besitzen. RIAs bieten z.B. Funktionalitäten und Benutzerschnittstellen an, die vor dem Web 2.0 nur in Desktopanwendungen vorzufinden waren.

Eine Möglichkeit zur Entwicklung von RIAs ist durch Ajax gegeben [OpenAjaxAlliance (2006), S.5; O'Reilly (2005)]. Ajax ist ein Designansatz zur Entwicklung von RIAs, die in allen populären Webbrowsern laufen und ausschließlich native Browsertechnologien verwenden [OpenAjaxAlliance (2006), Garrett (2005)]. In Ajax-Anwendungen werden heute unter anderem die Browsertechnologien JavaScript, HTML, CSS, DOM und XMLHttpRequest verwendet.

Einen speziellen Ansatz zur Entwicklung von Ajax-Anwendungen stellt das Google Web Toolkit (GWT)[1] dar [Google (2006)]. Mit dem GWT werden Ajax-Anwendungen in Java programmiert, getestet und debugged. Mittels eines Compilers wird die fertige Java-Anwendung dann in eine Ajax-Anwendung übersetzt.

Bisher gibt es nur sehr wenige öffentliche Erfahrungen mit dem GWT:

- Es gibt nur sehr wenige GWT-Anwendungen.
- Es sind noch keine Bücher über das GWT erschienen.
- Internetdokumente zum GWT haben in der Regel Tutorial-Charakter, stellen also keine Bewertung des GWTs dar.

Mit Ajax bzw. dem GWT sollen desktopähnliche Anwendungen entwickelt werden können [OpenAjaxAlliance (2006)]. Die Mind-Mapping-Anwendung FreeMind [Müller u. a. (o.J.)] ist ein Beispiel für eine Desktopanwendung. Das Mind-Mapping ist eine grafische Aufzeichnungstechnik, welche die volle Bandbreite kortikaler Fähigkeiten, wie Wort, Bild, Zahl, Logik, Rhythmus, Farbe und räumliches Bewusstsein des menschlichen Gehirns nutzt [Buzan und

[1] In dieser Arbeit wird die Version 1.2.22 des Google Web Toolkits betrachtet.

Buzan (2006)]. In Abbildung 1.1 ist die Benutzerschnittstelle von FreeMind dargestellt. FreeMind stellt eine grafisch mittelmäßig anspruchsvolle 2D-Desktopanwendung dar.

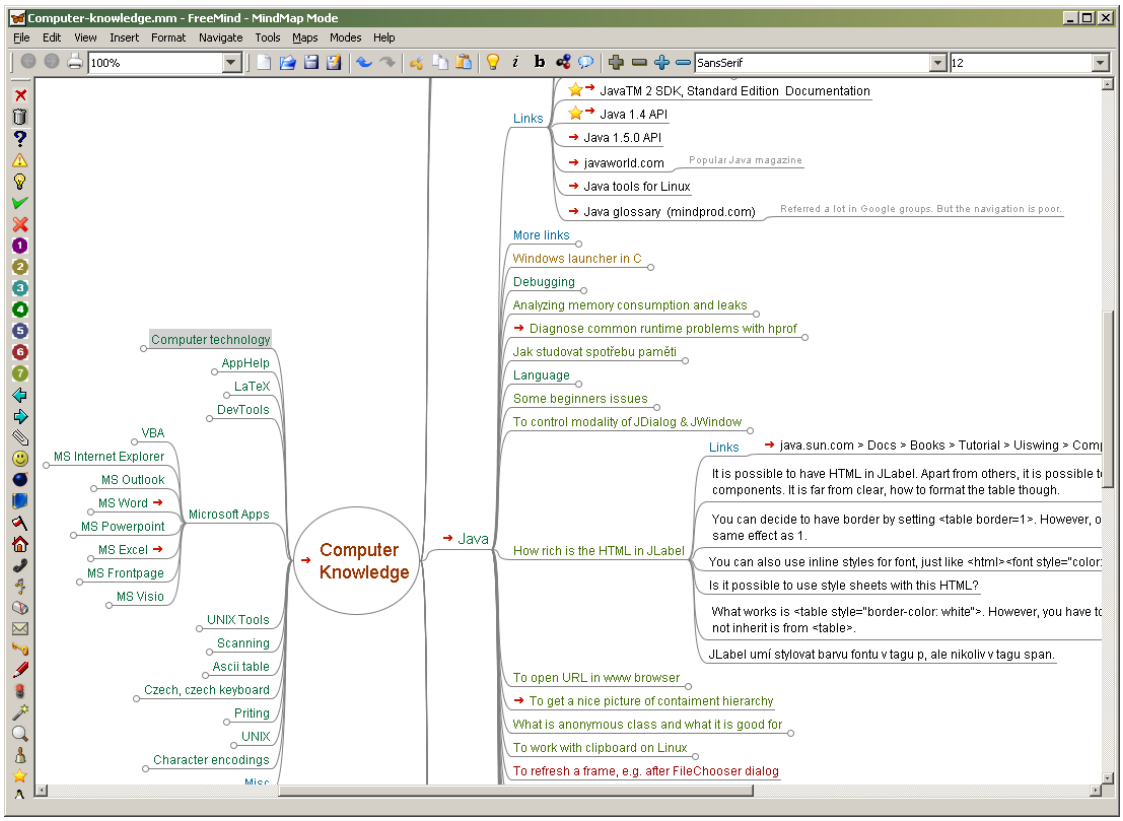

Abbildung 1.1: FreeMind [Müller u. a. (o.J.)]

1.1 Problem und Ziel

Alle die bestimmen müssen, ob das GWT zur Entwicklung grafisch mittelmäßig anspruchsvoller 2D-Ajax-Anwendungen geeignet ist, können aufgrund der mangelnden öffentlichen Erfahrungen mit dem GWT keine zuverlässigen Entscheidungen treffen. Das Ziel dieser Arbeit ist daher die Beantwortung der folgenden Fragen:

1. *Können mit dem Google Web Toolkit die Anforderungen an eine grafisch mittelmäßig anspruchsvolle 2D-Ajax-Anwendung umgesetzt werden? (Benutzersicht)*
2. *Können mit dem Google Web Toolkit 2D-Ajax-Anwendungen nach den State-of-the-Art-Entwurfsprinzipien gebaut werden? (Entwicklersicht)*

1.2 Vorgehen

Die Fragestellung dieser Arbeit soll durch die Entwicklung einer Beispielanwendung mit dem GWT beantwortet werden. Als Beispielanwendung soll eine Mind-Mapping-Anwendung entwickelt werden, weil Mind-Mapping-Anwendungen (z.B. FreeMind) grafisch mittelmäßig anspruchsvolle 2D-Anwendungen darstellen. Die zu entwickelnde Mind-Mapping-Anwendung wird in dieser Arbeit MiMa[2] genannt. FreeMind soll als Quelle der Anforderungen an MiMa dienen. MiMa soll einige grafisch mittelmäßig anspruchsvolle Anforderungen von FreeMind realisieren. Diese müssen nicht zu 100% mit denen von FreeMind übereinstimmen.

1.3 Kapitelübersicht

In den Kapiteln 2-4 werden einige Grundlagen, auf denen die Aufgabe dieser Arbeit basiert, beschrieben. Diese sind keine im Kontext der Aufgabenstellung erarbeiteten Beiträge des Autors, sondern komprimierte Darstellungen von Beiträgen anderer Autoren. Leser, die mit den Grundlagen bereits vertraut sind, können diese Kapitel getrost überspringen. Die Beschreibung der Grundlagen dient ausschließlich der Förderung des Verständnisses dieser Arbeit.

Ab Kapitel 5 beginnen die Beiträge des Autors. In den Kapiteln 5-7 wird die Entwicklung von MiMa beschrieben. Die Beantwortung der Fragestellung dieser Arbeit[3], d.h. die Evaluation des Google Web Toolkits, findet in Kapitel 7 zusammen mit der Beschreibung der Implementierung von MiMa statt.

Im Kapitel 8 werden die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick gegeben.

[2] Der Name MiMa wurde als Kurzschreibweise für Mind-Mapping gewählt
[3] siehe Abschnitt 1.1

2 Ajax

In diesem Kapitel wird beschrieben was Ajax ist. Zunächst wird Ajax aus der Sicht von Benutzern dargestellt. Es wird beschrieben, welche Anforderungen Ajax-Anwendungen erfüllen. Anschließend wird Ajax aus der Sicht von Entwicklern dargestellt. Es wird beschrieben, auf welche Weise Ajax diese Anforderungen erfüllt.

2.1 Benutzersicht

2.1.1 Desktopähnliche Benutzerschnittstellen

In der Einleitung wurde erwähnt, dass mit Ajax desktopähnliche Benutzerschnittstellen entwickelt werden können. Die Benutzerschnittstellen von Ajax-Anwendungen enthalten eine Vielzahl von Interaktionsmöglichkeiten, unter anderem Eingabefelder, Buttons, Listen, Menus, Bäume und Tabellen [Crane u. a. (2006)]. Die Benutzerschnittstellen sind in der Regel hochgradig interaktiv, z.B. bekommen die Benutzer nach Eingaben sofort Rückmeldungen oder können Features wie „Drag and Drop“ oder „Cut and Paste“ verwenden [Crane u. a. (2006)].

Die Interaktionen zwischen Benutzer und Anwendung können sehr flüssig ablaufen [Crane u. a. (2006), S.46]. Der Arbeitsfluss wird nicht wie bei HTML-Anwendungen jedes mal unterbrochen, wenn der Browser mit dem Server kommunizieren muss [Garrett (2005)]. Die Benutzerschnittstelle bleibt, wie es auch in Desktopanwendungen üblich ist, fast zu jedem Zeitpunkt bedienbar.

2.1.2 Plattformunabhängigkeit

Ajax-Anwendungen laufen in allen populären Browsern [OpenAjaxAlliance (2006), S.5]. Welche Browser heute als populär angesehen werden, ist nicht genau festgelegt. In Tabelle 2.1 ist der aktuelle Verbreitungsgrad verschiedener Browser dargestellt. In dieser Arbeit werden

die Browser Internet Explorer 6 (IE6) und Firefox (Fx)[1] als populär angesehen, weil diese zusammen schon einen Marktanteil von 79,8% ausmachen.

2006	IE7	IE6	IE5	Fx	Moz*	N7/8	O7/8/9
November	7.1%	49.9%	2.9%	29.9%	2.5%	0.2%	1.5%

Abbildung 2.1: Browserstatistik [W3Schools (o.J.)]

2.1.3 Keine Installation

Zur Ausführung von Ajax-Anwendungen muss der Benutzer keine Software installieren [Crane u. a. (2006), S.60]. Der Benutzer muss weder Browserplugins[2] noch Software auf dem Desktop installieren [OpenAjaxAlliance (2006), S.5]. Zur Ausführung von Ajax-Anwendungen wird nur ein populärer Browser (siehe Abschnitt 2.1.2) benötigt. Da heute jedes moderne Betriebssystem mit einem vorinstallierten Webbrowser geliefert wird, muss der Benutzer diesen nicht selbst installieren [Crane u. a. (2006), S.60].

2.2 Entwicklersicht

2.2.1 Ajax-Architektur

In diesem Abschnitt wird die grundlegende Architektur von Ajax-Anwendungen erläutert.

Interaktionsmodell

> *Ajax zeichnet sich nicht durch die Technologien aus, sondern vor allem durch das Interaktionsmodell, das Ajax durch Einsatz dieser Technologien bietet.* [Crane u. a. (2006), S.92]

In Abbildung 2.2 ist das Interaktionsmodell von Ajax-Anwendungen dargestellt.

[1] Da keine Versionsangabe für den Browser Firefox in der Tabelle 2.1 angegeben ist, wird davon ausgegangen, dass immer die neueste Version von Firefox gemeint ist

[2] Browserplugins müssen z.B. für Flash-Anwendungen oder Java-Applets installiert werden

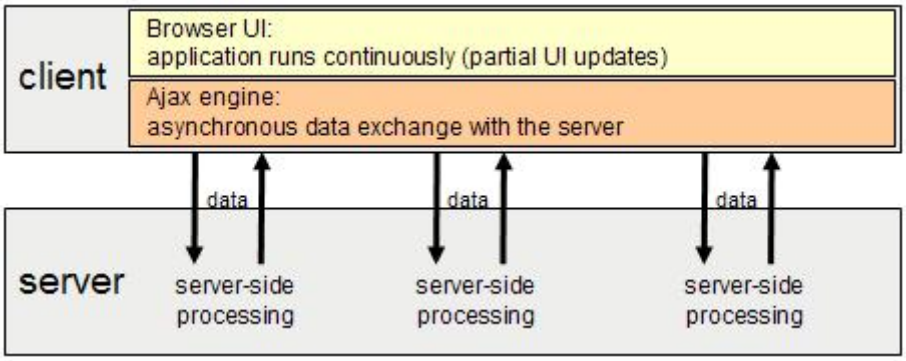

Abbildung 2.2: Ajax-Interaktionsmodell [OpenAjaxAlliance (2006)]

Ajax-Anwendungen basieren auf einem asynchronen Interaktionsmodell [Garrett (2005)]. In einem asynchronen Interaktionsmodell kann der Client sofort nach dem Abschicken einer Anfragenachricht weiterarbeiten, d.h. er blockiert nicht. Wenn für eine Anfragenachricht eine Antwortnachricht vom Server erwartet wird, kann diese zu jedem beliebigen Zeitpunkt vom Client empfangen werden. Da bei einem asynchronen Interaktionsmodell während der Kommunikation mit dem Server die Benutzerschnittstelle nicht blockiert, können dadurch hochgradig interaktive Benutzerschnittstellen entwickelt werden.

Das asynchrone Interaktionsmodell wird in Ajax-Anwendungen durch eine zusätzliche Schicht im Browser realisiert. Diese zusätzliche Schicht wird als Ajax-Engine bezeichnet.

Aktualisierung der Benutzerschnittstelle

Wenn der Client Anfragen an den Server schickt, werden in der Regel keine Seiten sondern Daten angefragt. Statt kompletter Seitenaktualisierungen, wie in klassischen HTML-Anwendungen, werden nur die Teile der Benutzerschnittstelle aktualisiert, die eine Aktualisierung benötigen.

Technologien

Um die Anforderung „Plattformunabhängigkeit“ (siehe 2.1.2) zu realisieren, werden in Ajax-Anwendungen nur von allen populären Browsern unterstützte Browsertechnologien verwendet [OpenAjaxAlliance (2006)].

Um die Anforderung „Keine Installation“ (siehe 2.1.3) zu realisieren, werden in Ajax-Anwendungen nur native Browsertechnologien verwendet [OpenAjaxAlliance (2006)]. Browserplugins sind keine nativen Browsertechnologien und dürfen daher nicht verwendet werden.

2.2.2 Kerntechnologien

In 2.2.1 wurde erwähnt, dass in Ajax-Anwendungen alle von populären Browsern nativ unterstützte Technologien verwendet werden können. In diesem Abschnitt werden nun die konkreten Kerntechnologien, die praktisch in jeder Ajax-Anwendung benutzt werden, beschrieben. Da diese Technologien heute wohlbekannt sind und auch unabhängig von Ajax verwendet werden können, wird nur ihre Rolle, die sie in Ajax-Anwendungen spielen, erläutert.

HTML

Mit HyperText Markup Language (HTML) werden die Struktur und die Widgets der Benutzerschnittstelle festgelegt. Weiterführende Informationen zu HTML finden sich z.B. in [W3C (2002)].

CSS

Mit Cascading Style Sheets (CSS) wird das Look-and-Feel (Erscheinungsbild) der Benutzerschnittstelle festgelegt. Es können z.B. Farben, Ränder, Hintergrundbilder, Größen und Anordnungen von Widgets definiert werden. Weiterführende Informationen zu CSS finden sich z.B. in [W3C (2006)].

DOM

Das Document Object Model (DOM) bietet eine API zur Manipulation und Abfrage des HTML-Dokuments an. Mit DOM kann die Benutzerschnittstelle dynamisch verändert werden. Es können z.B. neue Widgets hinzugefügt oder gerade angezeigte Widgets verändert werden. Weiterführende Informationen zum DOM finden sich z.B. in [W3C (2004)].

JavaScript

JavaScript ist eine objektorientierte Skriptsprache, die das Bindeglied zwischen HTML, CSS und DOM darstellt. Mit JavaScript wird die clientseitige Logik von Ajax-Anwendungen geschrieben. Der JavaScript-Code macht in der Regel einen großen Teil einer Ajax-Anwendung aus. Weiterführende Informationen zu JavaScript finden sich z.B. in [Flanagan (2002)].

XMLHttpRequest-Objekt

Die Technologien HTML, CSS, DOM und JavaScript werden auch als *Dynamic HTML (DHTML)* bezeichnet. Um 1997 war DHTML allgegenwärtig, konnte sich aber nicht durchsetzen. Solange eine Web-Anwendung nicht mit dem Server kommunizieren musste, konnten interaktive Benutzerschnittstellen entwickelt werden. Sobald aber mit dem Server kommuniziert werden musste, war aufgrund der synchronen Kommunikation und des Page-Refreshs die Interaktivität dahin. Die zur damaligen Zeit herrschenden Browser-Inkompatibilitäten taten ihr Übriges dazu. [Crane u. a. (2006), S.59]

Ajax stellt die fortgesetzte Evolution von DHTML dar [OpenAjaxAlliance (2006), S.4]. Der wesentliche Unterschied zwischen DHTML und Ajax ist das neue XMLHttpRequest-Objekt. Ajax könnte auch durch die Gleichung „Ajax = DHTML + XMLHttpRequest-Objekt“ beschrieben werden.

Das XMLHttpRequest-Objekt realisiert die Ajax-Engine in Ajax-Anwendungen und ist somit für die asynchrone Datenübertragung zuständig (siehe 2.2.1). Mit dem XMLHttpRequest-Objekt können beliebige Daten über HTTP übertragen werden. Als Datenformat können XML, JSON, HTML oder Plain Text zum Einsatz kommen. Weiterführende Informationen zum XMLHttpRequest-Objekt finden sich z.B. in [Crane u. a. (2006)].

2.2.3 Vektorgrafiken

Bei Grafiken kann man grundsätzlich zwischen Rastergrafiken und Vektorgrafiken unterscheiden. Rastergrafiken bestehen aus einer rasterförmigen Anordnung von sogenannten Pixeln (Bildpunkten), denen jeweils eine Farbe zugeordnet ist. Rastergrafiken werden vor allem zur Repräsentation komplexerer Bilder wie Fotos verwendet. Rastergrafiken werden im Web überwiegend in den Formaten gif, png und jpg bereitgestellt.

Bei Vektorgrafiken werden mathematische Verfahren zur Beschreibung eines Bildes verwendet. Eine Vektorgrafik wird aus grafischen Primitiven, wie Linien, Kreisen, Ellipsen, Rechtecken oder Polygonen, zusammengebaut. Um z.B. ein Kreis darzustellen werden nicht die einzelnen Pixel gespeichert, sondern nur der Mittelpunkt des Kreises und der Radius. Der

Kreis kann dann vom Darstellungsprogramm berechnet werden. Im Gegensatz zu Rastergrafiken können Vektorgrafiken beliebig skaliert werden, ohne einen Qualitätsverlust in der Darstellung zu erleiden.

Im Gegensatz zu Rastergrafiken werden Vektorgrafiken von Browsern noch nicht einheitlich unterstützt. In Tabelle 2.1 wird die aktuelle Unterstützung verschiedener Vektorgrafiktechnologien in den Browsern IE und Firefox dargestellt.[3] Desweiteren wird in Tabelle 2.1 dargestellt, ob die Vektorgrafiktechnologien das Zeichnen von Linien, Rechtecken, Kreisen und Bézierkurven unterstützen. Die Unterstützung anderer grafischer Primitive spielt in dieser Arbeit keine Rolle.

	IE	**Firefox**	**Linie**	**Rechteck**	**Kreis**	**Bézierkurve**
SVG	Plugin	Nativ	Ja	Ja	Ja	Ja
VML	Nativ	Nein	Ja	Ja	Ja	Ja
Canvas	ExplorerCanvas	Nativ	Ja	Ja	Ja	Ja
jsGraphics	Nativ	Nativ	Ja	Ja	Ja	Nein

Tabelle 2.1: Vektorgrafiktechnologien

Scalable Vector Graphics (SVG)

Scalable Vector Graphics (SVG) ist ein Standard des W3C [W3C (2003)]. SVG wird vom Firefox nativ unterstützt. Für den IE gibt es ein Plugin von Adobe. SVG unterstützt das Zeichnen von Linien, Rechtecken, Kreisen und Bézierkurven.

Vector Markup Language (VML)

Vector Markup Language (VML) wurde 1998 von Microsoft, Macromedia und Anderen zur Standardisierung beim W3C eingereicht [W3C (1998)]. VML wurde aber als Web Standard abgelehnt. VML wird vom IE nativ unterstützt. Firefox unterstützt VML nicht, auch nicht über ein Plugin. VML unterstützt das Zeichnen von Linien, Rechtecken, Kreisen und Bézierkurven.

[3] Andere Browser werden nicht dargestellt, weil in dieser Arbeit nur die populären Browser betrachtet werden (Siehe Abschnitt 2.1.2)

Canvas

Canvas erweitert HTML um das Element <canvas>. Das <canvas>-Element ist ein Teil der WhatWG Web applications 1.0 Spezifikation, auch bekannt als HTML 5 [WHATWG (2007)]. Canvas unterstützt das Zeichnen von Linien, Rechtecken, Kreisen und Bézierkurven. Canvas wird von Firefox nativ unterstützt. Für den IE gibt es die JavaScript-Bibliothek ExplorerCanvas, welche Canvas im IE emuliert. Die Emulation wird mit VML realisiert.

jsGraphics

jsGraphics ist eine von Walter Zorn entwickelte DHTML-basierte Vektorgrafikbibliothek [Zorn (o.J.)]. Da diese ausschließlich auf DHTML basiert, wird diese sowohl vom IE als auch von Firefox nativ unterstützt. Die aktuelle Version unterstützt das Zeichnen von Linien, Rechtecken und Kreisen, jedoch nicht das Zeichnen von Bézierkurven.

3 Google Web Toolkit

Das Schreiben von Ajax-Anwendungen ist heute ein umständlicher und fehleranfälliger Prozess [Google (2006)]. Man verbringt 90% der Entwicklungszeit damit, die subtilen Inkompatiblitäten zwischen den Browsern und Plattformen zu beherrschen [Google (2006)]. Ein weiteres Problem ist JavaScripts Mangel an Modularität, welches die Weitergabe, das Testen und Wiederverwenden von Ajax-Komponenten schwierig und zerbrechlich macht [Google (2006)].

Im Mai 2006 veröffentlichte Google das Google Web Toolkit (GWT). Mittlerweile ist das GWT vollständig open-source. Das GWT soll für Entwickler, welche die Browsersprache nicht sprechen, das Schreiben von Ajax-Anwendungen erleichtern [Google (2006)]. Mit dem GWT werden Ajax-Anwendungen in der Java-Umgebung entwickelt. Der gesamte Entwicklungszyklus, das Programmieren, das Debuggen und das Testen findet in Java statt. Läuft die Anwendung fehlerfrei, so lässt sie sich anschließend mittels des GWT-Compilers in eine browserkonforme Ajax-Anwendung übersetzen. Ein Java-basierter Ansatz ist sehr vielversprechend, weil [Google (2006)]

- es eine Vielzahl an mächtigen Java-Entwicklungswerkzeugen (z.B. Eclipse, IntelliJ, JProfiler, JUnit) gibt.
- das statische Typisierungskonzept von Java die Produktivität erhöht und die Fehler reduziert
- allgemeine Fehler (z.B. Tippfehler, Typisierungsfehler) schon zur Übersetzungszeit und nicht, wie in JavaScript, erst zur Laufzeit abgefangen werden.
- Features, wie Code-Completion oder automatisiertes Refaktorisieren, zur Verfügung stehen.
- Java-basierte OO-Designs einfacher als JavaScript-basierte Designs zu vermitteln und zu verstehen sind.

In den folgenden Unterabschnitten werden die Grundlagen zur Entwicklung von Benutzerschnittstellen mit dem GWT beschrieben. Da in dieser Arbeit die Entwicklung objektorientierter Benutzerschnittstellen im Vordergrund steht, werden unter anderem die folgenden Features des GWTs nicht beschrieben:

- Serverkommunikation

- Internationalisierung
- Browser-History-Management

3.1 GWT-Compiler

Das Herz des GWTs ist ein Compiler, der Java-Quellcode nach JavaScript übersetzt. Der Compiler kann die Kernsemantiken der Java-Sprache 1.4.2 oder früher übersetzten. Neue Java-Sprachfeatures, die erst ab Java 1.5 zur Verfügung stehen (z.B. Annotations, Generics oder Enumerations), können in GWT-Anwendungen nicht verwendet werden. Die folgenden Sprachfeatures (Semantiken) werden unter anderem nicht unterstützt oder nicht mit der gleichen Semantik nach JavaScript übersetzt:

- Der primitive Datentyp `long`: In JavaScript gibt es keinen 64-Bit Integer-Typ
- Multithreading und Synchronisation: JavaScript-Interpreter sind single-threaded
- Reflection

Das GWT unterstützt eine sehr kleine Untermenge der Laufzeitbibliothek von Java 1.4.2. Im Wesentlichen werden die Pakete `java.lang` und `java.util` unterstützt. Aber auch innerhalb dieser Pakete werden nicht alle Klassen und Methoden von Java unterstützt.

Der GWT-Compiler ist ein optimierender Compiler. Der resultierende JavaScript-Code ist nicht auf Lesbarkeit oder Wartbarkeit sondern auf Performanz getrimmt. Es wird ein monolitisches Skript generiert. Der resultierende Code ist „obfuscated“.

3.2 Hosted Mode und Web Mode

GWT-Anwendungen können in zwei Modi, dem *Hosted Mode* oder dem *Web Mode*, ausgeführt werden. Im Hosted Mode wird die Anwendung als Java-Bytecode in der Java Virtual Machine (JVM) ausgeführt. Im Web Mode wird die Anwendung als pures JavaScript und HTML in einem Webbrowser ausgeführt.

Um GWT-Anwendungen im Hosted Mode auszuführen, ist im GWT der *Hosted Webbrowser* enthalten. Der Hosted Webbrowser ist ein spezieller Browser, der sich in die JVM einhackt. Durch den Hosted Mode ist es möglich, GWT-Anwendungen in Java zu debuggen.

3.3 GUI-Framework

Das GWT beinhaltet ein GUI-Framework, welches große Ähnlichkeiten zu anderen Java-GUI-Frameworks (z.B. Swing oder SWT) besitzt [Google (2006)].

3.3.1 Widgets

Das GWT stellt eine Vielzahl generischer Widgets zur Verfügung. Auf [Google (2006)] werden die verfügbaren Widgets grafisch dargestellt.

Es lassen sich zwei Gruppen von Widgets unterscheiden: Steuerungselemente und Container. Steuerungselemente sind Widgets, wie Buttons, Labels oder Trees. Container sind Widgets, welche andere Widgets aufnehmen können. Container können sowohl Steuerungselemente als auch Container aufnehmen. Es gibt verschiedene Container-Typen. Diese Container-Typen ordnen die enthaltenen Widgets nach unterschiedlichen Strategien an. Es gibt z.B. Container-Typen, welche die Widgets

- horizontal in einer einzigen Zeile anordnen (`HorizontalPanel`)
- vertikal in einer einzigen Spalte anordnen (`VerticalPanel`)
- absolut positionieren (`AbsolutePanel`)

3.3.2 Benutzerdefinierte Widgets

Mit dem GWT können auch benutzerspezifische Widgets erstellt werden. Es gibt zwei Möglichkeiten benutzerspezifische Widgets zu erstellen.

Neue Widgets können am effektivsten durch Zusammensetzung (Komposition) bereits vorhandener Widgets erstellt werden [Google (2006)]. Das GWT stellt dazu die spezielle Widget-Klasse `Composite` zur Verfügung. Ein `Composite` nimmt ein anderes Widget auf (typischerweise einen Container) und verhält sich so wie das in ihm enthaltene Widget. Zusammengesetzte Widgets könnten auch durch Ableitung einer Container-Klasse erstellt werden. Die Benutzung des `Composite`s ist der Ableitung einer Container-Klasse vorzuziehen, weil ein Widget üblicherweise kontrollieren will, welche Methoden veröffentlicht werden sollen.

Neue Widgets können auch ohne Komposition erstellt werden. Mit dem GWT kann man DHTML in Java schreiben. Das GWT stellt dazu die Klasse `DOM` zur Verfügung. Ein neues Widget wird implementiert, indem die abstrakte Klasse `Widget` abgeleitet und in dieser das Widget mittels der Klasse `DOM` programmiert wird.

3.3.3 Erscheinungsbild

Das Erscheinungsbild (z.B. Farben, Ränder) der Widgets wird mittels CSS festgelegt. Jedes Widget besitzt einen Stilnamen, welches die Verbindung zwischen dem Widget und einer CSS-Regel herstellt. Der Stilname eines Widgets kann mittels der Methode `setStyleName()` gesetzt werden. Den im GWT enthalten Widgets sind schon Stilnamen zugewiesen. Ein Button besitzt z.B. den Stilnamen `gwt-Button`. Um das Erscheinungsbild eines Buttons festzulegen, würde man in der CSS-Datei z.B. die folgende CSS-Regel hinzufügen: `.gwt-Button {font-size: 150%;}`. Es gehört zum guten Stil, die Stilnamen der jeweiligen Widgets zu dokumentieren (z.B. in der Javadoc-Dokumentation). Wenn der Stilname an keiner Stelle dokumentiert wäre, müsste der Code zunächst durchforstet werden, um diesen zu finden.

3.3.4 Ereignisverarbeitung

Das Ereignis-Modell des GWT basiert auf dem bekannten „listener interface“ Model, welches z.B. auch in Swing verwendet wird. Widgets sind Ereignisquellen. Ein Button erzeugt z.B. ein Klickereignis, wenn dieser angeklickt wird. Ein Objekt, das im Falle eines Ereignisses benachrichtigt werden soll, um darauf zu reagieren, ist ein sogenannter Event-Listener. Eine Ereignisquelle stellt Methoden bereit, die es einem Listener erlauben, sich selbst der Liste interessierter Objekte hinzuzufügen oder sich aus ihr zu entfernen. An einem Button können sich z.B. ClickListener registrieren.

3.4 JavaScript Native Interface

Das GWT stellt eine an die „Java Native Interface“ (JNI) angelehnte Technik mit der Bezeichnung JavaScript Native Interface (JSNI) zur Verfügung, um JavaScript-Code zu integrieren. Manchmal kann es hilfreich sein, handgeschriebenes JavaScript in den Java-Source-Code zu integrieren. Einige Kernfunktionalitäten des GWTs basieren z.B. auf JSNI. Denkbar wäre auch die Integration von vorhandenen JavaScript-Bibliotheken, wie Prototype oder Script.aculo.us. Obwohl JSNI eine mächtige Technik ist, sollte diese aus den folgenden Gründen möglichst vermieden werden [Google (2006)]:

- Es besteht die Gefahr, plattformabhängigen Code zu schreiben.
- Es besteht die Gefahr, Speicherlöcher einzuführen.
- JSNI ist für Java-Tools weniger zugänglich.
- Die Optimierung durch den Compiler ist schwieriger.

3.5 JUnit-Integration

Bei der Programmierung mit Java werden Unit Tests häufig mit dem Test-Framework JUnit geschrieben. Testfälle werden in JUnit in eine von der Klasse `TestCase` abgeleiteten Klasse geschrieben. Das GWT beinhaltet die Unterklasse `GWTTestCase` von `TestCase`, welche die Benutzung von JUnit auch für GWT-Anwendungen ermöglicht. GWT-Anwendungen können sowohl im Hosted als auch im Web Mode getestet werden.

3.6 Vektorgrafiken

Das GWT bietet keine direkte Unterstützung für Vektorgrafiken. Das GWT kann aber durch die Einbindung externer Bibliotheken um Vektorgrafik-Funktionalität erweitert werden. Robert Hanson's „GWT Widget Library" enthält einige Widgets für SVG-basierte Vektorgrafiken und ein Wrapper-Widget für Walter Zorn's jsGraphics-Bibliothek [Hanson (o.J.)]. Alexei Sokolov stellt ein Widget für Canvas-basierte Vektorgrafiken zur Verfügung [Sokolov (o.J.)].

4 State-of-the-Art-Entwurfsprinzipien

In diesem Kapitel werden einige State-of-the-Art-Entwurfsprinzipien zur Entwicklung von Softwaresystemen im Allgemeinen und GUI-Anwendungen im Speziellen erläutert. Es werden nur die Entwurfsprinzipien erläutert, die für das weitere Verständnis dieser Arbeit notwendig sind und in dieser Arbeit benutzt wurden. Dieses Kapitel stellt also keineswegs eine umfassende Beschreibung gängiger Entwurfsprinzipien dar. Die Beschreibung der Entwurfsprinzipien fällt relativ kurz aus, weil diese wohlbekannt sind.

4.1 Entwurfsmuster

Flexible und erweiterbare Softwaresysteme weisen in der Regel eine Vielzahl von Entwurfsmustern auf.

> *Ein Entwurfsmuster benennt, motiviert und erläutert systematisch einen allgemeinen Entwurf, der ein in objektorientierten Systemen immer wiederkehrendes Entwurfsproblem löst.* [Gamma u. a. (1996), S.444]

In den folgenden Unterabschnitten werden die in dieser Arbeit verwendeten Entwurfsmuster beschrieben.

4.1.1 Schablonenmethode

Eine Schablonenmethode definiert das Skelett eines Algorithmus in einer Operation und delegiert die einzelnen Schritte an Unterklassen. Die Verwendung einer Schablonenmethode ermöglicht es Unterklassen, bestimmte Schritte eines Algorithmus zu überschreiben, ohne seine Struktur zu verändern [Gamma u. a. (1996), S.366-372].

Abbildung 4.1 zeigt die Struktur des Entwurfsmusters.

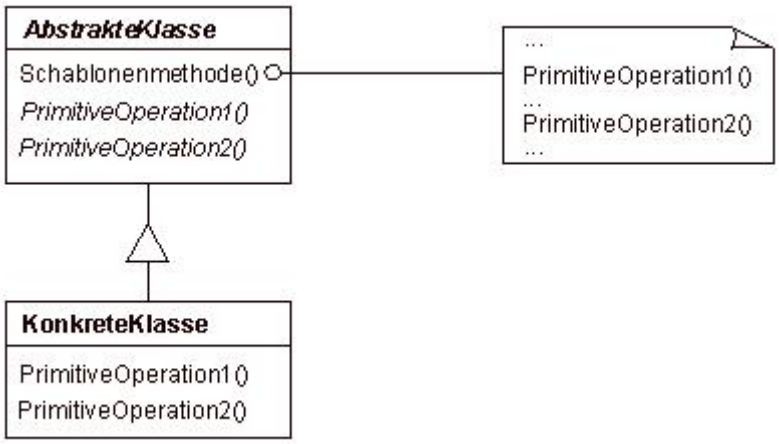

Abbildung 4.1: Schablonenmethode [Gamma u. a. (1996)]

AbstrakteKlasse ist in der Regel eine abstrakte Klasse. AbstrakteKlasse definiert primitive Operationen, welche die variablen Schritte des Algorithmus darstellen. Die primitiven Operationen können abstrakt oder konkret sein. Abstrakte primitive Operationen werden verwendet, wenn für die variablen Schritte des Algorithmus kein Standardverhalten angegeben werden kann. Andernfalls werden konkrete primitive Operationen verwendet. Konkrete primitive Operationen werden auch als Einschubmethoden bezeichnet. Des Weiteren können in einer Schablonenmethode auch konkrete Operationen, die nicht erst noch von Unterklassen definiert werden müssen, verwendet werden.

4.1.2 Beobachter

Der Zweck des Beobachtermusters wird in [Gamma u. a. (1996), S.287] folgendermaßen beschrieben:

> *Definiere eine 1-zu-N-Abhängigkeit zwischen Objekten, so dass die Änderung des Zustands eines Objekts dazu führt, dass alle abhängigen Objekte benachrichtigt und automatisch aktualisiert werden.*

Abbildung 4.2 zeigt die Struktur des Beobachtermusters.

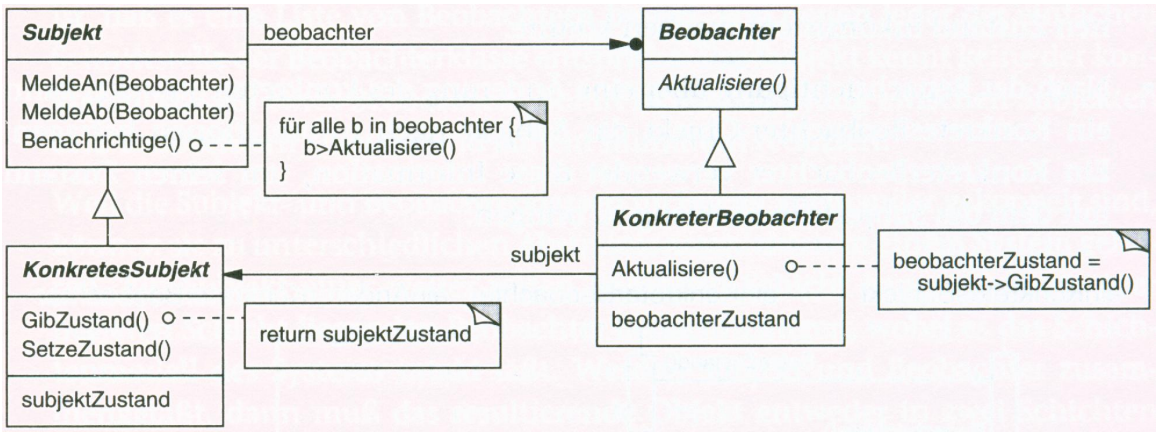

Abbildung 4.2: Beobachter [Gamma u. a. (1996)]

Das Objekt, von dem andere Objekte abhängen, wird Subjekt genannt. Die abhängigen Objekte werden Beobachter genannt. Ein Subjekt bietet eine Schnittstelle zum An- und Abmelden der Beobachter an. Ein Beobachter definiert eine Aktualisierungsschnittstelle. Wenn sich der Zustand des Subjekts ändert, benachrichtigt das Subjekt alle registrierten Beobachter. Die Beobachter können daraufhin ihren Zustand mit dem Subjektzustand in Einklang bringen.

4.1.3 Dekorierer

Das Dekorierermuster wird zur dynamischen Erweiterung der Zuständigkeiten eines Objekts verwendet. Dekorierer stellen eine flexible Alternative zur Unterklassenbildung dar, um die Funktionalität einer Klasse zu erweitern [Gamma u. a. (1996), S.199-211].

Abbildung 4.3 zeigt die Struktur des Dekorierermusters.

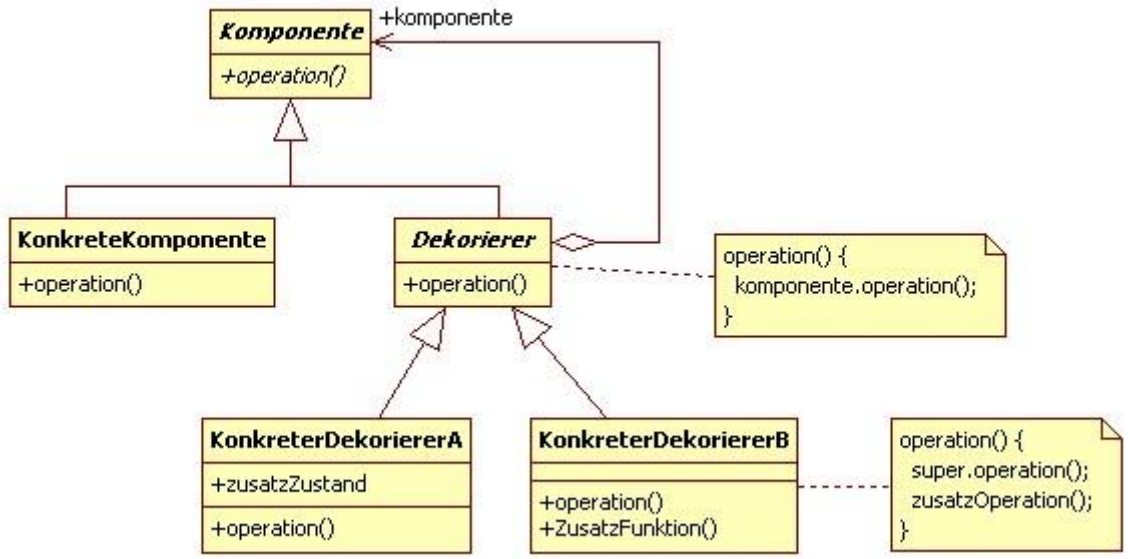

Abbildung 4.3: Dekorierer [Gamma u. a. (1996)]

KonkreteKomponente ist das Objekt, welches dynamisch um Zuständigkeiten erweitert werden kann. KonkreterDekorierer A und KonkreterDekorierer B stellen die möglichen Erweiterungen der Zuständigkeiten von KonkreteKomponente dar. Komponente definiert die gemeinsame Schnittstelle von KonkreteKomponente und Dekorierern. Die gemeinsame Schnittstelle ermöglicht es, dass Dekorierer überall dort verwendet werden können, wo es auch Komponente könnte und, dass Dekorierer rekursiv geschachtelt werden können. Ein Dekorierer leitet standardmäßig Anfragen einfach an sein Komponentenobjekt weiter. Konkrete Dekorierer erweitern dann nur die Operationen für die eine Zusatzfunktion ausgeführt werden soll.

4.2 Architekturmuster

4.2.1 Layers

Das Architekturmuster Layers wird z.B. in [Buschmann u. a. (2000), S.32] beschrieben. Layers wird häufig zur Zerlegung großer Systeme verwendet. Ein System wird dabei in übereinander liegende Schichten strukturiert (siehe Abbildung 4.4).

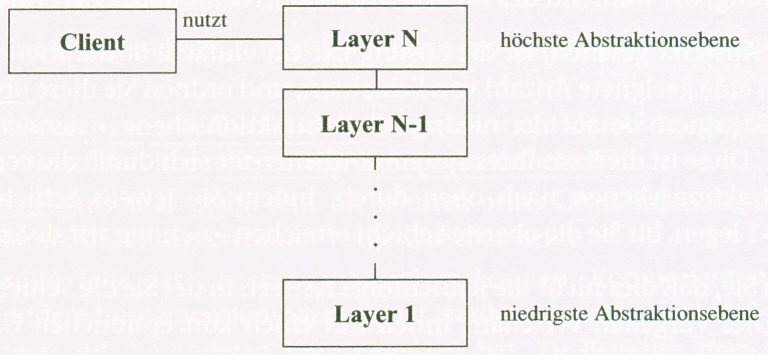

Abbildung 4.4: Das Architekturmuster Layers [Buschmann u. a. (2000)]

In einer Schicht werden Aufgaben der gleichen Abstraktionsebene behandelt. Wenn in einer Schicht Aufgaben verschiedener Abstraktionsebenen behandelt werden, ist dies eine Verletzung des Schichtenprinzips. Je niedriger eine Schicht ist, desto niedriger ist ihre Abstraktionsebene. Eine Schicht bietet der darüber liegenden Schicht ihre Dienste an. Um ihre Dienste bereitzustellen, nutzt sie selbst Dienste der darunter liegenden Schicht. Ein Grundprinzip des Layers-Musters ist, dass die Schicht N nur auf die Dienste der Schicht N-1 und nicht der Schicht N-2 zugreifen darf. Eine Schicht N darf natürlich auch die eigenen Dienste verwenden.

Die Schichten selbst werden häufig in weitere Komponenten zerlegt, man spricht dann auch von horizontaler Schichtung. Die Komponenten einer Schicht dürfen direkt miteinander sprechen.

Die Vorteile des Layers-Musters sind unter anderem [Buschmann u. a. (2000), S.50-51]:

- Wiederverwendung von Schichten
- Abhängigkeiten bleiben lokal: Bei notwendigen Änderungen muss in der Regel nur Code einer Schicht verändert werden.
- Austauschbarkeit: Schichten können komplett durch eine semantisch äquivalente Implementierung ausgetauscht werden.

Die Anwendung des Layers-Musters hat unter anderem die folgenden Nachteile [Buschmann u. a. (2000), S.51-53]:

- Kaskaden veränderten Verhaltens: „Das Schichtenprinzip wird zum Nachteil, wenn man eine auf den ersten Blick lokale Änderung auch in vielen anderen Schichten vornehmen muss.“
- Geringere Effizienz: Eine Schichtenarchitektur ist aufgrund der Indirektion in der Regel weniger effizient als eine monolithische Struktur.

- Unnötige Arbeit: „Wenn einige Dienste, die von niedrigeren Schichten erledigt werden, viele oder redundante Arbeiten durchführen, die nicht unbedingt von den oberen Schichten benötigt werden, nimmt die Performanz des Systems ab.“

4.2.2 Separated-Presentation

Das Architekturmuster Separated-Presentation wird in [Fowler (o.J.)] beschrieben. Separated-Presentation ist ein Speziallfall des Architekturmusters Layers [Fowler (o.J.)].

Bei Separated-Presentation wird ein System in eine Domänen- und eine Präsentationsschicht aufgeteilt. Die Domänenschicht enthält die Domänenlogik. Unter Domänenlogik wird die in der betrachteten Anwendungsdomäne zu verrichtende Arbeit verstanden [Fowler (2005), S.20]. Dies beinhaltet z.B. die Durchführung von Berechnungen und die Validierung von Daten.

Die Präsentationsschicht enthält die Präsentationslogik. Präsentationslogik definiert wie Interaktionen zwischen dem Benutzer und der Software gehandhabt werden sollen [Fowler (2005), S.19]. Unter Präsentationslogik fallen z.B. Kommandozeilenschnittstellen oder grafische Benutzerschnittstellen (GUIs). Die Hauptverantwortlichkeiten der Präsentationslogik sind die Darstellung der Domänenobjekte auf dem Bildschirm, und die Interaktionen des Benutzers als Aktionen auf die Domäne zu interpretieren.

Zwischen der Präsentations- und der Domänenschicht gibt es eine gerichtete Abhängigkeitsbeziehung. Die Präsentationsschicht darf auf die Domänenschicht zugreifen, umgekehrt darf die Domänenschicht aber nicht auf die Präsentationsschicht zugreifen.

Schichten sind logische und keine physikalischen Konstrukte, dennoch können Schichten auf verschiedene Prozesse verteilt werden. Z.B. läuft in klassischen HTML-Anwendungen die Domänenschicht auf einem Server und die Präsentationsschicht (zum Teil) im Webbrowser.

In Java werden Schichten häufig auf Java-Pakete physikalisch abgebildet [Fowler (o.J.)].

Da Separated-Presentation ein Speziallfall des Layers-Musters ist, gelten für Separated-Presentation die gleichen Vor- und Nachteile wie für das Layers-Muster. Interpretiert man die Vorteile des Layers-Musters im Kontext von Separated-Presentation, bedeutet dies, dass der Aufwand zur Entwicklung neuer Benutzerschnittstellen für ein System reduziert wird, weil die Domänenschicht wiederverwendet werden kann.

In Anwendungen, bei denen die Daten auch persistiert werden, wird die Domänenschicht häufig noch in eine Persistenzschicht aufgeteilt. Man spricht dann auch von einer 3-Schichtenarchitektur. In dieser Arbeit wird die Persistenzschicht nicht weiter betrachtet, weil keine Daten persistiert werden müssen.

5 Anforderungen

In diesem Kapitel werden die Anforderungen an MiMa beschrieben. Da in dieser Arbeit beurteilt werden soll, wie gut grafisch mittelmäßig anspruchsvolle 2D-Ajax-Anwendungen mit dem GWT entwickelt werden können, stehen die Anforderungen an die Benutzerschnittstelle von MiMa im Vordergrund. Bei den Anforderungen an die Benutzerschnittstelle wird zwischen Interaktions- und Darstellungsanforderungen unterschieden.

Die Mind-Mapping-Anwendung FreeMind dient als Quelle der Anforderungen. Es werden einige grafisch mittelmäßig anspruchsvolle Anforderungen von FreeMind für MiMa übernommen. Diese stimmen nicht zu 100% mit denen von FreeMind überein. Viele Anforderungen von FreeMind, wie z.B. das Speichern oder Exportieren von Mind-Maps, werden nicht betrachtet, weil in dieser Arbeit die Benutzerschnittstelle im Vordergrund steht.

5.1 FreeMind Mind-Maps

In Abbildung 5.1 ist eine Mind-Map, die mit FreeMind erstellt wurde, dargestellt. Sie zeigt die Wesentlichen Eigenschaften, die in FreeMind für Mind-Maps festgelegt werden können.

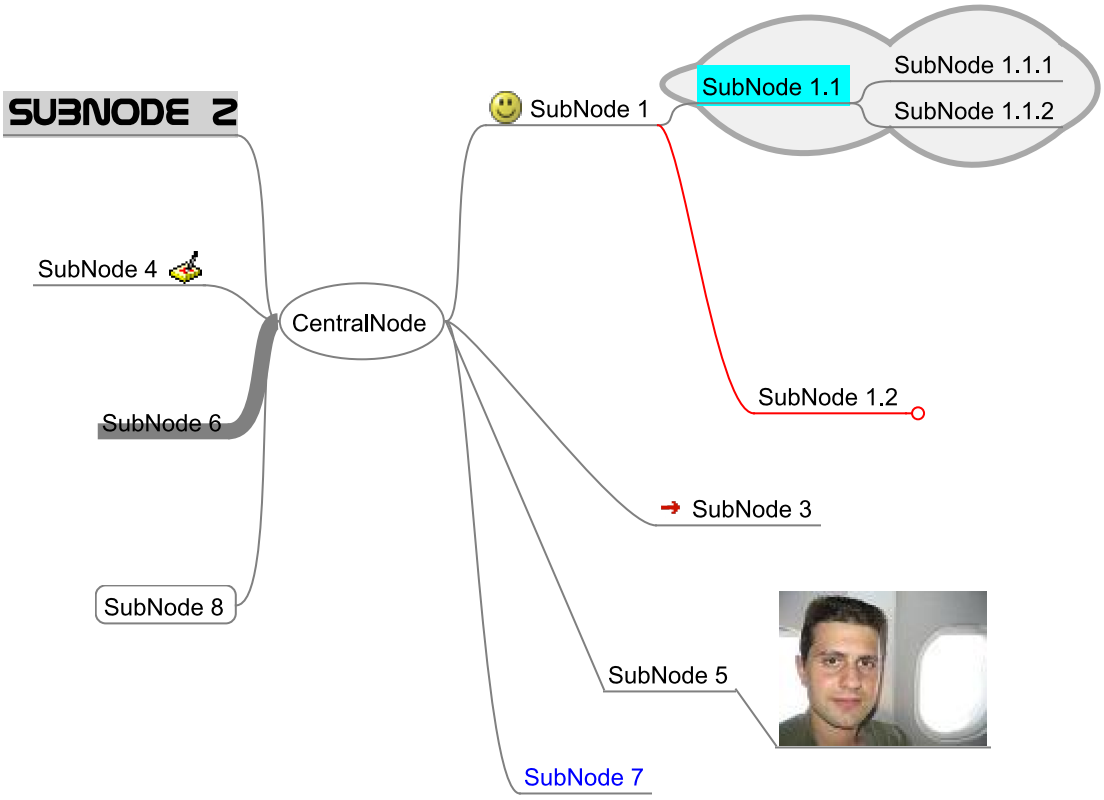

Abbildung 5.1: FreeMind Mind-Map

In FreeMind können für Mind-Maps z.B. folgende Dinge festgelegt werden:

- Kantenfarbe (z.B. besitzt „SubNode 1.2“ eine rote Kantenfarbe)
- Knotentitelfarbe (z.B. besitzt „SubNode 7“ eine blaue Knotentitelfarbe)
- Knotenhintergrundfarbe (z.B. besitzt „SubNode 1.1“ eine türkise Knotenhintergrundfarbe)
- Kantendicke (z.B. besitzt „SubNode 6“ eine dicke Kante)
- Kantenform (z.B. besitzt „SubNode 5“ eine lineare Kantenform)
- Schriftart (z.B. besitzt „SubNode 2“ eine spezielle Schriftart)
- Icons (z.B. besitzt „SubNode 1“ ein Smiley-Icon)
- Notiz (z.B. besitzt „SubNode 4“ eine Notiz, dies wird durch das Notiz-Icon angedeutet)
- Link (z.B. besitzt „SubNode 3“ einen Link auf „www.google.de“, dieses wird durch das Link-Icon angedeutet)
- Bilder (z.B. besitzt der Unterknoten von „SubNode 5“ ein Bild)

- Knotenrahmen (z.B. ist um den Knoten „SubNode 1.1“ mitsamt seiner Unterknoten eine Wolke gezeichnet)

5.2 Domänenmodell

In Abbildung 5.2 ist das zur Abbildung 5.1 entsprechende Domänenmodell dargestellt. Es ist nur der in dieser Arbeit betrachtete Ausschnitt des Domänenmodells von FreeMind dargestellt. Das Domänenmodell ist in Form eines UML-Klassendiagramms dargestellt. Die Mind-Map in Abbildung 5.1 stellt ein Exemplar bzw. eine konkrete Ausprägung des in Abbildung 5.2 dargestellten Domänenmodells dar.

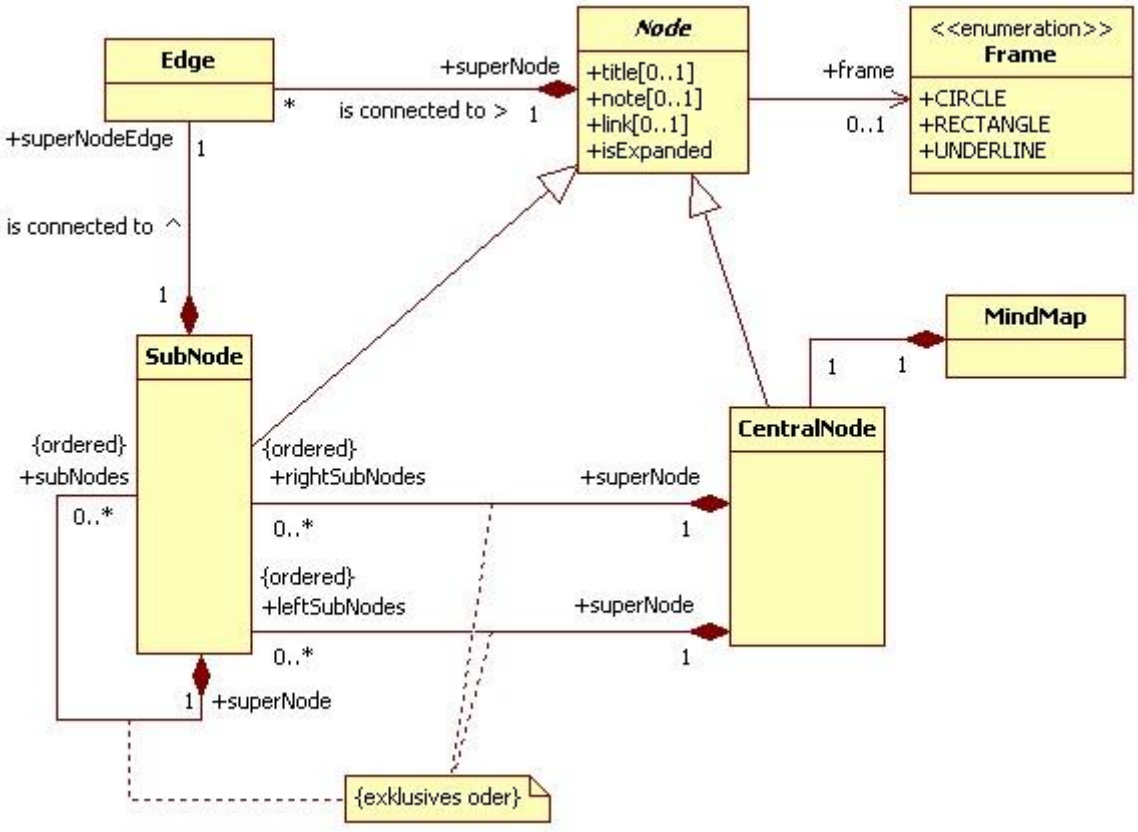

Abbildung 5.2: Domänenmodell

Eine Mind-Map ist nichts anderes als eine Baumstruktur. Eine Mind-Map besitzt immer ge-

nau einen Zentralknoten. Der Zentralknoten ist die Wurzel der Baumstruktur. Ein Zentralknoten kann beliebig viele linke und rechte Unterknoten besitzen. Sowohl die linken als auch die rechten Unterknoten sind geordnet. Ein Unterknoten hat genau einen Oberknoten. Ein Unterknoten kann beliebig viele weitere Unterknoten besitzen. Die Unterknoten sind geordnet. Die Notiz „exklusives oder“ in der Abbildung 5.2 bedeutet, dass genau eine der drei Beziehungen für einen Unterknoten gilt. Ein Unterknoten kann nicht zugleich Unterknoten eines Unterknotens und des Zentralknotens sein. Die Gemeinsamkeiten von Unterknoten und Zentralknoten sind in der abstrakten Klasse Knoten dargestellt. Die Klasse ist abstrakt, weil es in einer Mind-Map nur Knoten vom Typ Zentralknoten oder Unterknoten und nicht vom Typ Knoten geben kann. Ein Knoten kann einen Titel, einen Link und/oder eine Notiz besitzen. Alle Angaben sind optional. Ein Knoten ist entweder expandiert oder kollabiert. Jeder Unterknoten besitzt genau eine Kante zu seinem Oberknoten. Ein Knoten besitzt einen Kreisrahmen, Rechteckrahmen, Unterlinierahmen oder keinen Rahmen.

5.2.1 Linke und rechte Mind-Map-Hälften

Wenn ein Unterknoten zum Zentralknoten hinzugefügt wird, wird dieser entweder als rechter oder als linker Unterknoten hinzugefügt. Ein Unterknoten wird als rechter Unterknoten hinzugefügt, wenn die Anzahl der rechten Unterknoten kleiner oder gleich der Anzahl der linken Unterknoten ist. Andernfalls wird der Unterknoten als linker Unterknoten hinzugefügt.

Wenn ein Unterknoten des Zentralknotens entfernt wird, bleibt die Links-Rechts-Verteilung der restlichen Unterknoten erhalten. Die Anzahl der rechten und die der linken Unterknoten kann sich daher um mehr als eins unterscheiden.

5.3 Interaktion

In diesem Abschnitt werden die Interaktionsanforderungen an MiMa beschrieben. Die Interaktionsanforderungen werden in Form von Anwendungsfällen beschrieben, weil Anwendungsfälle die Interaktionen zwischen den Akteuren (Systemumgebung) und dem System beschreiben [Cockburn (2001)]. In dieser Arbeit übernimmt MiMa die Rolle des Systems und ein Benutzer die Rolle des Akteurs. Anwendungsfälle stellen zusätzlich das externe Systemverhalten in einer abgegrenzten Arbeitssituation dar. In den Unterabschnitten werden die einzelnen Anwendungsfälle in Tabellenform beschrieben. In den Tabellen werden die Benutzer mit einem B und MiMa mit einem M dargestellt.

5.3.1 Mind-Map erstellen

Schritt	Akteur	Beschreibung
1	B	gibt die URL von M in die Browseraddressleiste ein.
2	M	stellt den Zentralknoten im Zentrum des Browserfensters dar. Der Zentralknoten besitzt den Titel „Edit this node“ und besitzt einen Kreisrahmen.

Tabelle 5.1: Erfolgsszenario „Mind-Map erstellen“

5.3.2 Titel bearbeiten

Schritt	Akteur	Beschreibung
1	B	klickt mit rechter Maustaste auf den Knoten dessen Titel bearbeitet werden soll.
2	M	zeigt Popup-Menu über dem angeklickten Knoten an.
3	B	klickt auf Menu-Eintrag „Edit Title“.
4	M	zeigt an Stelle des Knotens ein Texteingabefeld an.
5	B	schreibt den Titel in das Texteingabefeld und bestätigt mit der Taste „Enter“.
6	M	zeigt den eingegebenen Titel für den bearbeiteten Knoten an.

Tabelle 5.2: Erfolgsszenario „Titel bearbeiten“

Schritt	Bedingung	Beschreibung
5a	B hat die Taste „Escape“ gedrückt	M zeigt die Mind-Map, so wie sie vor Ausführung des Anwendungsfalls dargestellt wurde, an.

Tabelle 5.3: Erweiterungsszenario „Titel bearbeiten“

5.3.3 Unterknoten erstellen

Schritt	Akteur	Beschreibung
1	B	klickt mit rechter Maustaste auf den Knoten von dem ein Unterknoten erstellt werden soll.
2	M	zeigt Popup-Menu über dem angeklickten Knoten an.
3	B	klickt auf Menu-Eintrag „Create Subnode“.
4	M	expandiert den Knoten und zeigt an der späteren Position des zu erstellenden Unterknotens ein Texteingabefeld an. Es wird eine Kante, so wie in Abschnitt 5.4.1 beschrieben, zwischen dem angeklickten Knoten und dem Texteingabefeld gezeichnet. Der Positionierungsalgorithmus wird in Abschnitt 5.2.1 und 5.4.2 beschrieben.
5	B	schreibt den Titel für den zu erstellenden Unterknoten in das Texteingabefeld und bestätigt mit der Taste „Enter“.
6	M	zeigt den neuen Unterknoten mit dem eingegebenen Titel an. Der Unterknoten besitzt einen Unterlinierahmen.

Tabelle 5.4: Erfolgsszenario „Unterknoten erstellen“

Für Schritt 5 gilt das Erweiterungsszenario aus Tabelle 5.3.

5.3.4 Unterknoten entfernen

Dieser Anwendungsfall wird als „Unterknoten entfernen“ und nicht allgemeiner als „Knoten entfernen“ bezeichnet, weil der Zentralknoten nicht entfernt werden kann.

Schritt	Akteur	Beschreibung
1	B	klickt mit rechter Maustaste auf den zu entfernenden Unterknoten.
2	M	zeigt Popup-Menu über dem angeklickten Unterknoten an.
3	B	klickt auf Menu-Eintrag „Cut Node“.
4	M	entfernt den Unterknoten mitsamt seiner Unterknoten.

Tabelle 5.5: Erfolgsszenario „Unterknoten entfernen“

5.3.5 Unterknoten umbewegen

Um einen Unterknoten mitsamt seiner Unterknoten umzubewegen, führt der Benutzer zunächst den Anwendungsfall „Unterknoten entfernen“ für den umzubewegenden Unterknoten aus. Anschließend werden die in Tabelle 5.6 beschriebenen Schritte ausgeführt.

Schritt	Akteur	Beschreibung
1	B	klickt mit rechter Maustaste auf den Knoten, an dem der entfernte Unterknoten angefügt werden soll.
2	M	zeigt Popup-Menu über dem angeklickten Knoten an.
3	B	klickt auf Menu-Eintrag „Paste Node“.
4	M	expandiert den Knoten und fügt den Unterknoten an den Knoten an.

Tabelle 5.6: Erfolgsszenario „Unterknoten umbewegen“

5.3.6 Knoten expandieren/kollabieren

Schritt	Akteur	Beschreibung
1	B	klickt mit linker Maustaste auf den zu expandierenden/kollabierenden Knoten.
2	M	expandiert den Knoten, wenn dieser kollabiert ist. M kollabiert den Knoten, wenn dieser expandiert ist.

Tabelle 5.7: Erfolgsszenario „Knoten expandieren/kollabieren“

5.3.7 Knoten vollständig expandieren

Schritt	Akteur	Beschreibung
1	B	klickt mit rechter Maustaste auf den vollständig zu expandierenden Knoten.
2	M	zeigt Popup-Menu über dem angeklickten Knoten an.
3	B	klickt auf Menu-Eintrag „Expand All Subnodes“.
4	M	expandiert die gesamte Knotenhierarchie des zu expandierenden Knotens.

Tabelle 5.8: Erfolgsszenario „Knoten vollständig expandieren“

5.3.8 Rahmen festlegen

Schritt	Akteur	Beschreibung
1	B	klickt mit rechter Maustaste auf den Knoten, dessen Rahmen festgelegt werden soll.
2	M	zeigt Popup-Menu über dem angeklickten Knoten an.
3	B	klickt auf den Menu-Eintrag „Set Rectangle Frame“, „Set Circle Frame“, „Set Underline Frame“ oder „Set No Frame“.
4	M	legt den zum gewählten Menu-Eintrag entsprechenden Rahmen für den Knoten fest.

Tabelle 5.9: Erfolgsszenario „Rahmen festlegen“

5.3.9 Link erstellen

Schritt	Akteur	Beschreibung
1	B	klickt mit rechter Maustaste auf dem Knoten, für den ein Link erstellt werden soll.
2	M	zeigt Popup-Menu über den angeklickten Knoten an.
3	B	klickt auf Menu-Eintrag „Set Link“.
4	M	zeigt die Link-Bearbeitungsdialogbox aus Abbildung 5.3 über dem Knoten an.
5	B	gibt den Link in das Texteingabefeld ein und drückt anschließend die Schaltfläche „Accept“.
6	M	übernimmt den eingegebenen Link für den Knoten.

Tabelle 5.10: Erfolgsszenario „Link erstellen“

Schritt	Bedingung	Beschreibung
5a	B hat die Schaltfläche „Cancel“ gedrückt	M zeigt die Mind-Map, so wie sie vor Ausführung des Anwendungsfalls dargestellt wurde, an.

Tabelle 5.11: Erweiterungsszenario „Link erstellen“

Abbildung 5.3: Link-Bearbeitungsdialogbox

5.3.10 Link folgen

Schritt	Akteur	Beschreibung
1	B	klickt auf das Link-Icon eines Knotens.
2	M	zeigt die Ressource, auf die der Link zeigt, in einem neuen Browserfenster an.

Tabelle 5.12: Erfolgsszenario „Link folgen“

5.3.11 Link entfernen

Dieser Anwendungsfall kann nur ausgeführt werden, wenn der Knoten, dessen Link entfernt werden soll, einen Link besitzt.

Schritt	Akteur	Beschreibung
1	B	klickt mit der rechten Maustaste auf den Knoten, dessen Link entfernt werden soll.
2	M	zeigt Popup-Menu über dem angeklickten Knoten an.
3	B	klickt auf Menu-Eintrag „Remove Link“.
4	M	entfernt den Link des bearbeiteten Knotens.

Tabelle 5.13: Erfolgsszenario „Link entfernen“

5.3.12 Notiz erstellen

Schritt	Akteur	Beschreibung
1	B	klickt mit rechter Maustaste auf den Knoten, für den eine Notiz erstellt werden soll.
2	M	zeigt Popup-Menu über dem angeklickten Knoten an.
3	B	klickt auf Menu-Eintrag „Set Note“.
4	M	zeigt die Notiz-Bearbeitungsdialogbox aus Abbildung 5.4 über dem Knoten an.
5	B	gibt die Notiz in das Texteingabefeld ein und drückt anschließend die Schaltfläche „Accept“.
6	M	übernimmt die eingegebene Notiz für den Knoten.

Tabelle 5.14: Erfolgsszenario „Notiz erstellen“

Für Schritt 5 gilt das Erweiterungsszenario aus Tabelle 5.11.

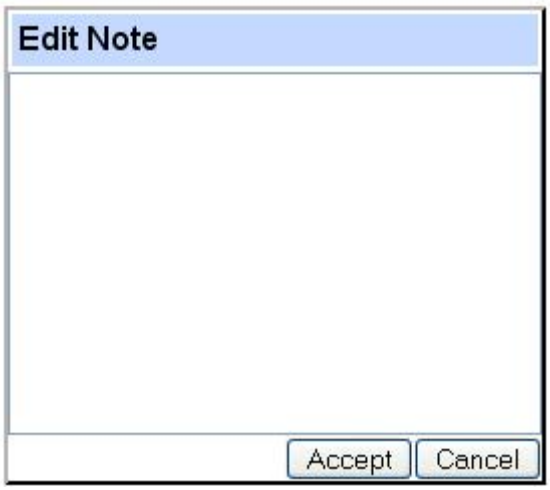

Abbildung 5.4: Notiz-Bearbeitungsdialogbox

5.3.13 Notiz anzeigen

Schritt	Akteur	Beschreibung
1	B	klickt auf das Notiz-Icon eines Knotens.
2	M	zeigt die Notiz des Knotens in der Notiz-Anzeigedialogbox aus Abbildung 5.5 an.
3	B	liest die Notiz und klickt anschließend auf einen freien Bereich außerhalb der Notiz-Anzeigedialogbox.
4	M	schließt die Notiz-Anzeigedialogbox.

Tabelle 5.15: Erfolgsszenario „Notiz anzeigen“

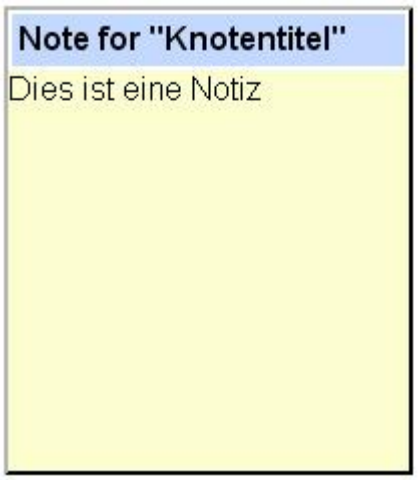

Abbildung 5.5: Notiz-Anzeigedialogbox

5.3.14 Notiz entfernen

Dieser Anwendungsfall kann nur ausgeführt werden, wenn der Knoten, dessen Notiz entfernt werden soll, eine Notiz besitzt.

Schritt	Akteur	Beschreibung
1	B	klickt mit der rechten Maustaste auf den Knoten, dessen Notiz entfernt werden soll.
2	M	zeigt Popup-Menu über dem angeklickten Knoten an.
3	B	klickt auf Menu-Eintrag „Remove Note“.
4	M	entfernt die Notiz des Knotens.

Tabelle 5.16: Erfolgsszenario „Notiz entfernen“

5.4 Darstellung

5.4.1 Knoten

In Abbildung 5.6 werden die Elemente eines Knotens (Zentral- und Unterknotens) dargestellt und bezeichnet.

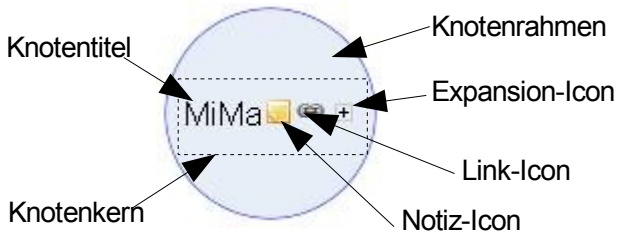

Abbildung 5.6: Elemente eines Knotens

Knotenrahmen

Bei Knoten wird zwischen Knotenrahmen und Knotenkern unterschieden. Der Knotenkern beinhaltet die Elemente Knotentitel, Link-Icon, Notiz-Icon und Expansion-Icon. Ein Knotenrahmen stellt eine Verzierung des Knotenkerns dar bzw. umgibt den Knotenkern. Der Knotenrahmen passt sich automatisch der Größe des Knotenkerns an. Z.B. vergrößert sich der Knotenrahmen, wenn der Knotentitel länger wird.

Knoten können als Knotenrahmen einen Kreisrahmen, einen Rechteckrahmen, einen Unterlinierahmen oder keinen Rahmen besitzen.

Ein Kreisrahmen soll in hellblau[1] dargestellt werden. Der Kreisrand soll blau[2] sein. Die Abstände zwischen Kreisrand und Knotenansicht sollen mindestens 20 Pixel betragen.

Ein Rechteckrahmen soll in gelb[3] dargestellt werden. Der Rechteckrand soll in grün[4] dargestellt werden.

Ein Unterlinierahmen unterschreicht die Knotenansicht. Die Unterlinie soll in blau dargestellt werden.

Icons

Ein Knoten zeigt ein Expansion-Icon an, wenn der Knoten Unterknoten besitzt. Andernfalls wird kein Expansion-Icon angezeigt. Ein Expansion-Icon zeigt an, ob der Knoten expandiert[5] oder kollabiert[6] ist. Im expandierten Zustand wird ein Collapse-Icon angezeigt, im kollabierten Zustand ein Expand-Icon[7].

Für Knoten, die eine Notiz besitzen, wird ein Notiz-Icon angezeigt. Für Knoten, die einen Link besitzen, wird ein Link-Icon angezeigt.

Das Expansion-Icon wird in linken Unterknoten ganz links in der Knotenansicht angeordnet. In rechten Unterknoten wird dieses ganz rechts angeordnet. Das Link-Icon und das Notiz-Icon wird immer rechts neben dem Knotentitel angeordnet, aber vor dem Expansion-Icon.

Hintergrundfarbe

Wenn der Mauszeiger über einem Element des Knotenkerns (z.B. Link-Icon) verharrt, wird die Hintergrundfarbe des Elements in hellgrau[8] dargestellt. Wenn der Mauszeiger den Bereich des Elements wieder verlässt, wird die ursprüngliche Hintergrundfarbe angezeigt.

Kanten

Jeder Unterknoten ist mit seinem Oberknoten durch eine Kante verbunden. Die Kanten werden als Bézierkurven dargestellt. Sie sollen in blauer Farbe dargestellt werden.

[1] Genauer soll die RGB-Farbe #E8EEF7 (hexadezimale Form) verwendet werden.
[2] Genauer soll die RGB-Farbe #0000FF (hexadezimale Form) verwendet werden.
[3] Genauer soll die RGB-Farbe #FFFF00 (hexadezimale Form) verwendet werden.
[4] Genauer soll die RGB-Farbe #00FF00 (hexadezimale Form) verwendet werden.
[5] Im expandierten Zustand eines Knotens sind dessen Unterknoten sichtbar.
[6] Im kollabierten Zustand eines Knotens sind dessen Unterknoten nicht sichtbar.
[7] In Abbildung 5.6 wird als Expansion-Icon ein Expand-Icon angezeigt
[8] Genauer soll die RGB-Farbe #DDDDDD (hexadezimale Form) verwendet werden.

5.4.2 Radiale Mind-Map-Struktur

In Abbildung 5.7 ist dargestellt, wie Unterknoten beim Hinzufügen positioniert werden. Es ist nur die rechte Seite des Zentralknotens dargestellt, weil die Positionierung der Unterknoten auf der linken und auf der rechten Seite des Zentralknotens gleich funktioniert. Zur Übersicht sind die Kanten zwischen den Knoten weggelassen. Die Kästen A und B sind nur zur Veranschaulichung des Positionierungsalgorithmus der Knoten dargestellt. Diese werden in der echten Mind-Map nicht dargestellt.

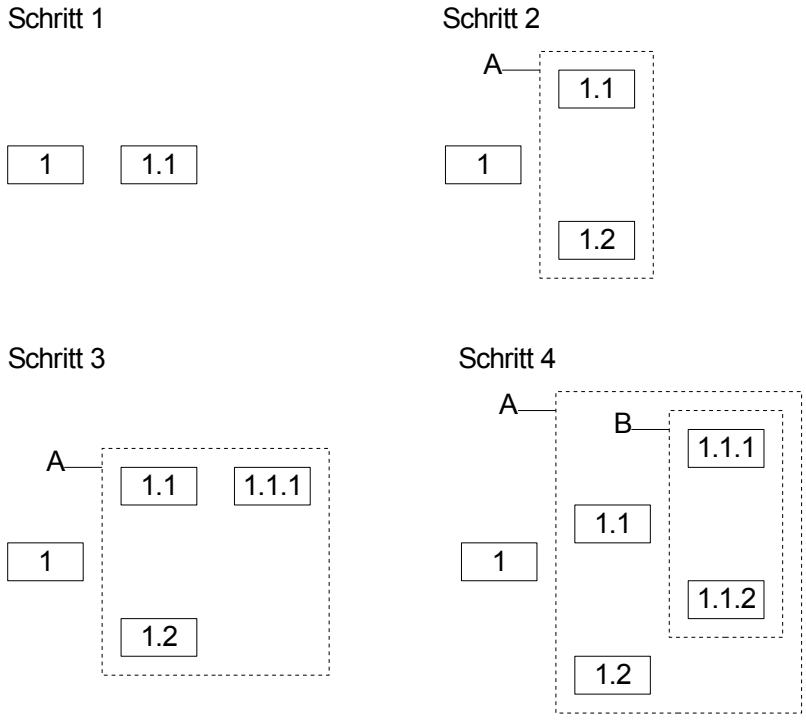

Abbildung 5.7: Radiale Mind-Map-Struktur

In Schritt 1 wird der Unterknoten 1.1 dem Knoten 1 hinzugefügt. Der Unterknoten 1.1 ist am Knoten 1 zentriert ausgerichtet. In Schritt 2 wird der Unterknoten 1.2 dem Knoten 1 hinzugefügt. Der imaginäre Kasten A ist am Knoten 1 zentriert ausgerichtet. In Schritt 3 wird der Unterknoten 1.1.1 dem Knoten 1.1 hinzugefügt. Der Unterknoten 1.1.1 ist am Knoten 1.1 zentriert ausgerichtet. Der imaginäre Kasten A ist am Knoten 1 zentriert ausgerichtet. In Schritt 4 wird der Unterknoten 1.1.2 dem Knoten 1.1 hinzugefügt. Der imaginäre Kasten B ist am Knoten 1.1 zentriert ausgerichtet. Der imaginäre Kasten A ist am Knoten 1 zentriert ausgerichtet.

Wenn ein Unterknoten entfernt, expandiert oder kollabiert wird, wird die Mind-Map erneut ausgerichtet. Wenn z.B. in Schritt 4 der Unterknoten 1.1.2 entfernt oder kollabiert wird, wird die Mind-Map wie in Schritt 3 ausgerichtet.

5.5 Technische Anforderungen

5.5.1 Änderbare Benutzerschnittstelle

Die Benutzerschnittstellen von Anwendungen werden im Gegensatz zum funktionalen Kern viel häufiger geändert und angepasst [Buschmann u. a. (2000), S.122]. In den folgenden Unterabschnitten werden verschiedene Aspekte von Benutzerschnittstellen, die sich im Lebenslauf einer interaktiven Anwendung häufig verändern, beschrieben. Für die verschiedenen Aspekte werden die entsprechenden Anforderungen an das System beschrieben.

Neue Benutzerschnittstellen

Häufig ist es im Lebenslauf einer interaktiven Anwendung notwendig die Benutzerschnittstelle um eine neue zu erweitern oder durch eine andere zu ersetzen. In dieser Arbeit soll nur eine Benutzerschnittstelle entwickelt werden. Es soll eine GWT-basierte Benutzerschnittstelle bzw. allgemeiner eine Web-Benutzerschnittstelle entwickelt werden. In Zukunft könnte sich vielleicht herausstellen, dass eine GWT-basierte Benutzerschnittstelle für eine Mind-Mapping-Anwendung nicht so geeignet ist. Vielleicht könnten die Grafik-Möglichkeiten des GWTs bzw. eines Webbrowsers den Anforderungen einer Mind-Mapping-Anwendung nicht genügen. In diesem Fall wäre es denkbar, dass statt einer GWT-basierten Web-Anwendung und Benutzerschnittstelle eine Desktop-Anwendung mit Swing-basierter Benutzerschnittstelle entwickelt wird. Für das zu entwickelnde System sollen daher neue Benutzerschnittstellen mit möglichst wenig Aufwand entwickelt werden können.

Änderbares Look-and-Feel

In GUI-Anwendungen ändert sich häufig auch das Look-and-Feel (Erscheinungsbild) der Benutzerschnittstelle. Das Look-and-Feel soll daher mit möglichst wenig Aufwand verändert werden können.

5.5.2 Abhängigkeiten zum GWT minimieren

Die Benutzung des GWTs ist eine Framework-Entscheidung. Framework-Entscheidungen sind meist sehr risikobehaftet, weil diese später oft nur unter sehr großem Aufwand rückgängig gemacht werden können. Um das Risiko einer falschen Entscheidung zu minimieren, ist es daher ratsam, die Abhängigkeiten zum GWT möglichst gering zu halten. Wenn das GWT später ausgetauscht werden soll, sollte dies auf andere Komponenten möglichst wenig Auswirkungen haben.

5.6 Fazit

In diesem Kapitel wurden die Anforderungen an MiMa beschrieben. Anforderungen, wie kreisförmige Knotenrahmen, Kanten in Form von Bézierkurven oder die radiale Mind-Map-Struktur, zeigen, dass MiMa eine grafisch mittelmäßig anspruchsvolle 2D-Ajax-Anwendung ist.

Um die Frage

> *Können mit dem GWT die Anforderungen an eine grafisch mittelmäßig anspruchsvolle 2D-Ajax-Anwendung umgesetzt werden?*

aus dem Einleitungskapitel zu beantworten, sollten als nächstes die Anforderungen aus diesem Kapitel mit dem GWT realisiert werden.

6 Systementwurf

In dieser Arbeit soll nicht nur beurteilt werden, ob mit dem GWT eine grafisch mittelmäßig anspruchsvolle 2D-Ajax-Anwendung überhaupt realisiert werden kann, sondern auch ob diese nach den State-of-the-Art-Entwurfsprinzipien entwickelt werden kann. In diesem Kapitel wird ein Systementwurf für die Anforderungen aus Kapitel 5 vorgestellt. Der Systementwurf wird nach den State-of-the-Art-Entwurfsprinzipien erstellt.

FreeMind ist in der Programmiersprache Java implementiert und damit ein objektorientiertes System. Aus diesem Grund wird für MiMa ein objektorientierter und nicht z.B. prozeduraler Systementwurf erstellt. MiMa wird mittels der „Unified Modeling Language“ (UML) entworfen, weil diese die Standardsprache für objektorientierte Systementwicklung ist.

MiMa besitzt eine Schichtenarchitektur. Die Aufteilung des Systems in Schichten und die dahinterstehende Motivation wird in Abschnitt 6.1 beschrieben. Die einzelnen Schichten werden in den darauffolgenden Abschnitten erläutert.

6.1 Schichtenarchitektur

6.1.1 Separated-Presentation

MiMa wird sowohl Domänen- als auch Präsentationslogik enthalten. Z.B. ist folgendes Domänenlogik:

> *Ein Unterknoten wird als rechter Unterknoten hinzugefügt, wenn die Anzahl der rechten Unterknoten kleiner oder gleich der Anzahl der linken Unterknoten ist. Andernfalls wird der Unterknoten als linker Unterknoten hinzugefügt.* (siehe Unterabschnitt 5.2.1)

Präsentationslogik ist z.B.:

> *Für Knoten, die eine Notiz besitzen, wird ein Notiz-Icon angezeigt. Für Knoten, die einen Link besitzen, wird ein Link-Icon angezeigt.* (siehe Unterabschnitt 5.4.1)

MiMa wird nach dem Architekturmuster Separated-Presentation[1] organisiert. Durch Separated-Presentation wird die Anforderung „Neue Benutzerschnittstellen“[2] erfüllt. Wenn eine neue Benutzerschnittstelle entwickelt werden soll, kann die Domänenschicht wiederverwendet werden. Gäbe es keine strikte Trennung zwischen Präsentations- und Domänenlogik, müsste bei der Entwicklung einer neuen Benutzerschnittstelle auch die Domänenlogik neu programmiert werden. Der Aufwand zur Entwicklung einer neuen Benutzerschnittstelle wird durch die Aufteilung des Systems in eine Präsentations- und eine Domänenschicht reduziert. Die Präsentationsschicht von MiMa wird im Folgenden `presentation` und die Domänenschicht `domain` genannt.

6.1.2 Schichten

Es ist sinnvoll, die Schichten `presentation` und `domain` in weitere Schichten zu zerlegen. In Abbildung 6.1 sind alle Schichten von MiMa und die Abhängigkeitsbeziehungen zwischen den Schichten dargestellt.

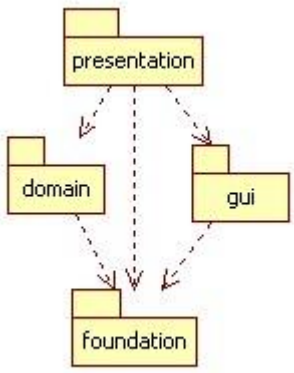

Abbildung 6.1: Schichten

In der Regel werden bei der Anwendungsentwicklung grundlegende Klassen, wie `Collection`, `Integer`, `Double` oder `String`, benötigt. Alle Schichten werden solche Basisklassen benötigen. Basisklassen werden in der Schicht `foundation` angesiedelt.

Die Schicht `gui` enthält hochgradig wiederverwendbare Benutzerschnittstellenklassen. In dieser Schicht werden Klassen angesiedelt, die potentiell auch für die Entwicklung anderer GUI-basierter Anwendungen verwendet werden können. Z.B. könnte diese Schicht Klassen

[1] siehe Unterabschnitt 4.2.2
[2] siehe Unterabschnitt 5.5.1

für Eingabefelder, Menus oder Schaltflächen beinhalten, weil diese in Benutzerschnittstellen häufig verwendet werden.

6.1.3 Physikalische Schichtung und Verteilung

FreeMind ist keine verteilte Anwendung. Eine funktionsbedingte Verteilung von MiMa ist daher nicht notwendig. Eine funktionsbedingte Verteilung wäre z.B. notwendig, wenn Daten von verschiedenen Benutzern gemeinsam genutzt werden sollen oder allgemeiner eine gemeinsame Ressourcennutzung gewünscht ist. Eine Verteilung von MiMa könnte z.B. aufgrund von nicht-funktionalen Anforderungen, z.B. Sicherheit oder Performanz, dennoch sinnvoll sein. Da in dieser Arbeit die Benutzerschnittstelle im Vordergrund steht, wird keine verteilte Anwendung entwickelt, auch wenn eine Verteilung unter Umständen die bessere Lösung wäre. Aus diesem Grund werden alle Schichten von MiMa im Webbrowser und nicht auf einem Server ausgeführt. Durch eine klare logische Schichtung sollte eine spätere Verteilung der Schichten auf Client und Server dennoch mit wenig Aufwand möglich sein.

6.2 Schicht „foundation“

In Abbildung 6.2 sind die Komponenten der Schicht `foundation` dargestellt. Diese Komponenten sind typischerweise Bestandteil jeder Programmierumgebung und werden daher nicht genauer spezifiziert.

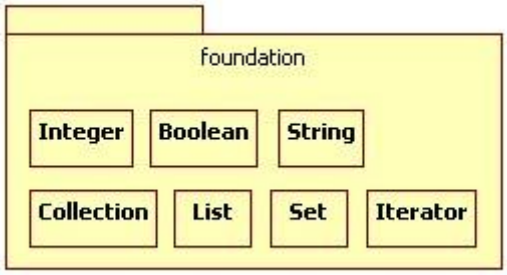

Abbildung 6.2: Basisschicht

6.3 Schicht „domain“

In diesem Abschnitt wird der Systementwurf für die Schicht `domain` beschrieben.

6.3.1 Organisation

Es gibt verschiedene Möglichkeiten, eine Domänenschicht zu organisieren. In [Fowler (2003)] werden die Architekturmuster

- Transaction-Script: „Organizes business logic by procedures where each procedure handles a single request from the presentation.“ [Fowler (2003), S. 110]
- Domain-Model: „An object model of the domain that incorporates both behavior and data.“ [Fowler (2003), S.116]
- Table-Module: „A single instance that handles the business logic for all rows in a database table or view.“ [Fowler (2003), S.125]

zur Strukturierung der Domänenschicht beschrieben.

Die Domänenschicht `domain` wird nach dem Architekturmuster Domain-Model organisiert. Domain-Model wird Transaction-Script vorgezogen, weil Transaction-Script ein stark prozeduraler Ansatz, d.h. nicht objektorientierter Ansatz, ist. Das Table-Module Architekturmuster macht für MiMa keinen Sinn, weil für MiMa keine Persistenzschicht entwickelt werden soll.

6.3.2 Schnittstelle

In diesem Abschnitt wird die Schnittstelle zur Schicht `domain` beschrieben. Die Schnittstelle repräsentiert die Dienste, welche die Schicht `domain` der Schicht `presentation` anbietet. In Abbildung 6.3 ist die Schnittstelle der Schicht `domain` als UML-Klassendiagramm dargestellt.

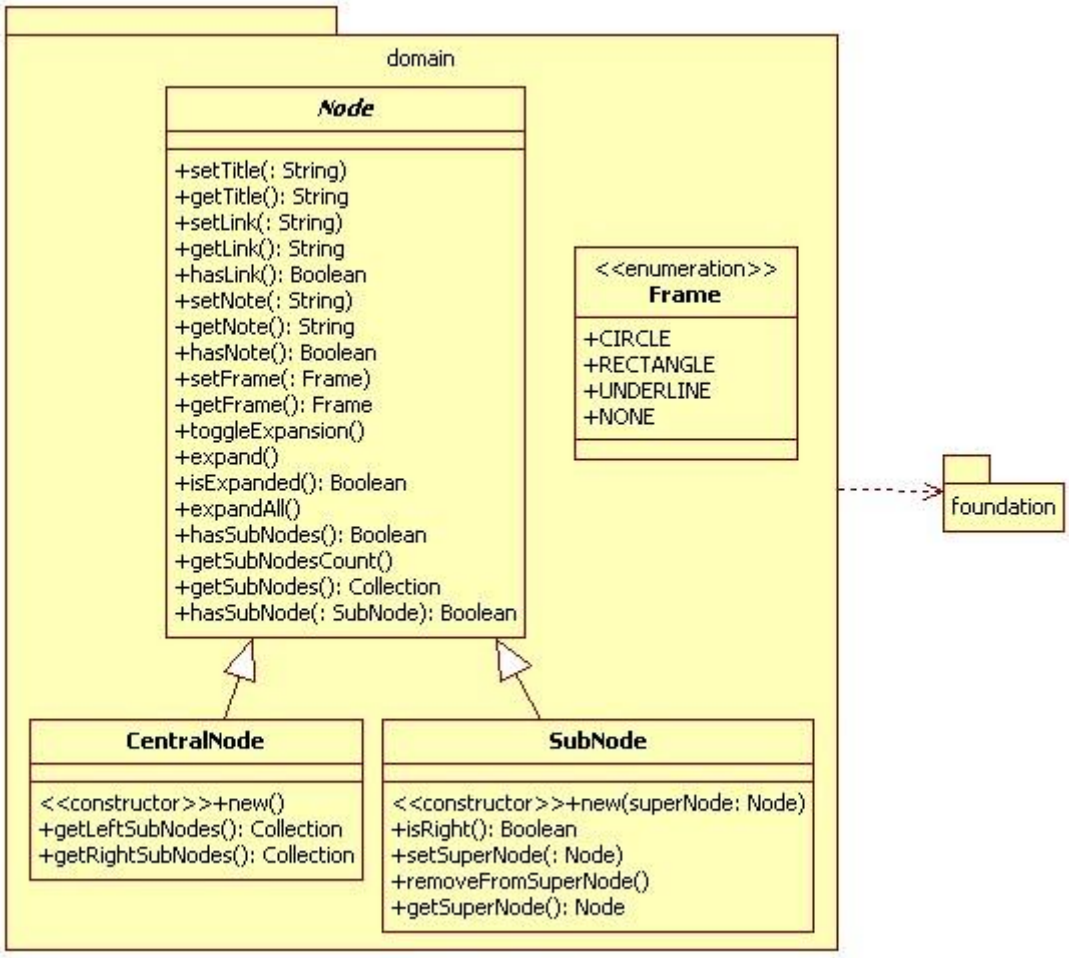

Abbildung 6.3: Schnittstelle zur Domänenschicht

Da die Schicht `domain` nach dem Architekturmuster Domain-Model organisiert wird, kann das Domänenmodell aus Abschnitt 5.2 als Ausgangspunkt zur Definition der Schnittstelle verwendet werden. Es ist sinnvoll, im Systementwurf möglichst die gleichen Begriffe wie im Anwendungsbereich zu verwenden, weil dies das System verständlicher macht.

Eine ausführliche Diskussion der Schnittstelle findet nicht statt. Stattdessen wird in Tabelle 6.1 nur der Kontext, in dem die Operationen benötigt werden, beschrieben.

Operation	Kontext
`setTitle(:String)`	Anwendungsfall „Titel bearbeiten“ (siehe 5.3.2)
`getTitle():String`	Darstellung des Knotentitels (siehe 5.4.1)
`setLink(:String)`	Anwendungsfall „Link erstellen“ und „Link entfernen“ (siehe 5.3.9 und 5.3.11)
`getLink(:String)`	Anwendungsfall „Link folgen“ (siehe 5.3.10)
`hasLink():Boolean`	Darstellung des Link-Icons (siehe 5.4.1)
`setNote(:String)`	Anwendungsfall „Notiz erstellen“ und „Notiz entfernen“ (siehe 5.3.12 und 5.3.14)
`getNote(:String)`	Anwendungsfall „Notiz anzeigen“ (siehe 5.3.13)
`hasNote():Boolean`	Darstellung des Link-Icons (siehe 5.4.1)
`setFrame(:Frame)`	Anwendungsfall „Rahmen festlegen“ (siehe 5.3.8)
`getFrame():Frame`	Darstellung des Knotenrahmens (siehe 5.4.1)
`toggleExpansion()`	Anwendungsfall „Knoten expandieren/kollabieren (siehe 5.3.6)
`expand()`	Anwendungsfall „Unterknoten erstellen“ und „Unterknoten umbewegen“ (siehe 5.3.3 und 5.3.5)
`isExpanded():Boolean`	Darstellung des Expansion-Icons (siehe 5.4.1)
`expandAll()`	Anwendungsfall „Knoten vollständig expandieren“ (siehe 5.3.7)
`hasSubNodes():Boolean`	Darstellung des Expansion-Icons (siehe 5.4.1)
`getLeftSubNodes():List`	Darstellung der linken Unterknoten des Zentralknotens
`getRightSubNodes():List`	Darstellung der rechten Unterknoten des Zentralknotens
`isRight():Boolean`	Darstellung der Kanten und des Expansion-Icons (siehe 5.4.1 und 5.4.1)
`setSuperNode(:Node)`	Anwendungsfall „Unterknoten umbewegen“ (siehe 5.3.5)
`removeFromSuperNode()`	Anwendungsfall „Unterknoten entfernen“ (siehe 5.3.4)
`getSubNodes():List`	Darstellung der Unterknoten eines Unterknotens

Tabelle 6.1: Operationen und Kontext

Einige Operationen der Schnittstelle wurden in Tabelle 6.1 nicht beschrieben, weil diese von der Schicht `presentation` nicht benötigt werden. Diese werden aktuell nur innerhalb der Schicht `domain` selbst verwendet.

6.3.3 Realisierung

In diesem Unterabschnitt wird der Systementwurf für die Realisierung der Schnittstelle der Schicht `domain` beschrieben. Es werden nur einige Teilaspekte der Realisierung besprochen, weil in dieser Arbeit die Benutzerschnittstelle im Vordergrund steht.

Statischer Systementwurf

In Abbildung 6.4 ist der statische Systementwurf für die Realisierung der Schnittstelle in Form eines UML-Klassendiagramms dargestellt.

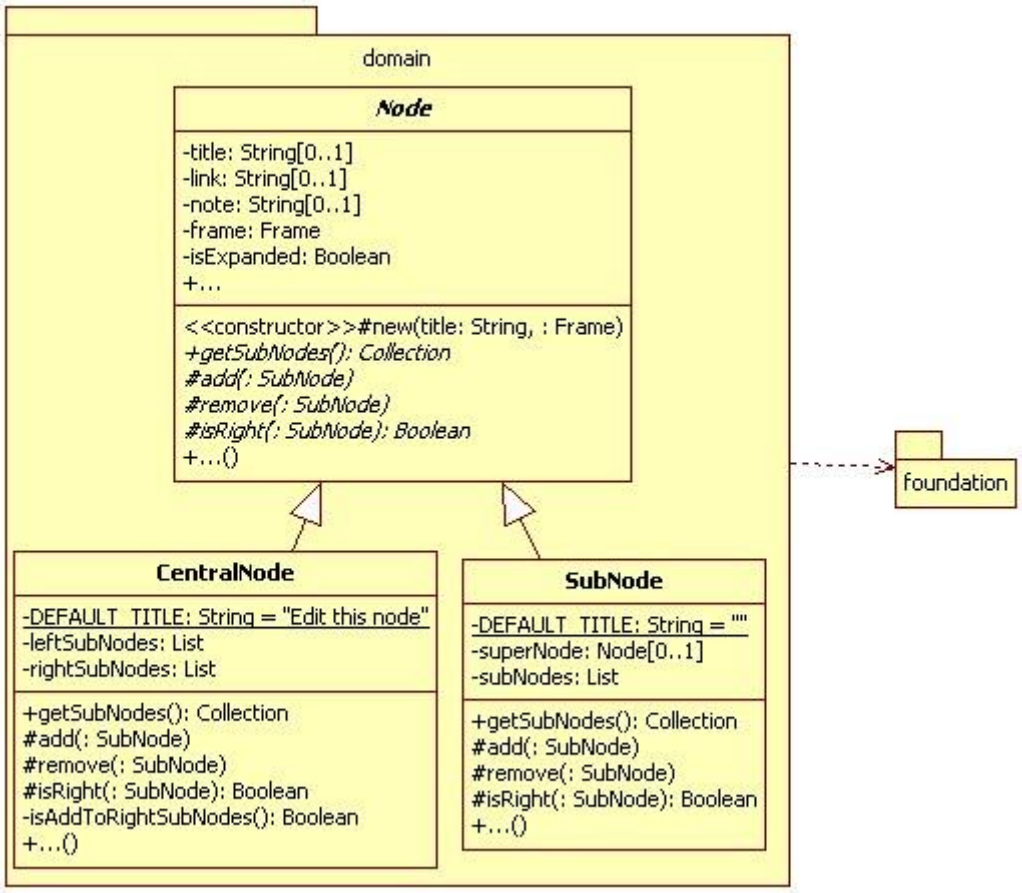

Abbildung 6.4: Statische Realisierung der Domänenschicht

Die Attribute von `Node` und `CentralNode` sind selbsterklärend und werden daher nicht weiter erläutert. Das Attribut `superNode` in der Klasse `SubNode` stellt eine Verbindung zum aktuellen Oberknoten her. Auf konzeptioneller Ebene besitzt jeder Unterknoten genau einen Oberknoten.[3] Auf dem Bildschirm sind z.B. zu keinem Zeitpunkt frei schwebende Unterknoten erlaubt. Um den Anwendungsfall „Unterknoten umbewegen“ zu unterstützen, ist im Systementwurf hingegen die Angabe eines Oberknotens optional. Wenn auf einem Unterknoten die „Cut“-Operation ausgeführt wird, wird der Unterknoten von seinem Oberknoten entfernt. Wenn anschließend die „Paste“-Operation ausgeführt wird, wird dem Unterknoten wieder ein Oberknoten zugewiesen.

Die Operationen `add(:SubNode)`, `remove(:SubNode)` und `isRight(:SubNode)` sind geschützt (protected), weil diese nur intern von `SubNode` und `CentralNode` zur

[3] siehe Abschnitt 5.2

Realisierung ihrer öffentlichen Schnittstelle verwendet werden dürfen. Diese Operationen sind abstrakt, weil nur die Unterklassen wissen, wie die Operationen durchzuführen sind. Ein `Node` weiß z.B. nicht, wie ein Unterknoten entfernt werden kann. `CentralNode` und `SubNode` hingegen wissen, wie Unterknoten zu entfernen sind, weil diese ihre Unterknoten in den Listen `leftSubNodes`, `rightSubNodes` und `subNodes` verwalten.

Die Operation `expandAll()` der Klasse `Node` ist eine rekursive Schablonenmethode. Das Entwurfsmuster Schablonenmethode ist in Abschnitt 4.1.1 beschrieben. Die Operation `expandAll()` ist die eigentliche Schablonenmethode, d.h. diese definiert das Skelett des Algorithmus. `expandAll()` ruft die konkrete primitive Operation `expand()` und die abstrakte primitive Operation `getSubNodes():Collection` auf. Das dynamische Verhalten von `expandAll()` wird im Folgenden noch erläutert.

Im Folgenden wird nur der Systementwurf für die Realisierung der Operationen `expandAll()` und `setSuperNode(:Node)` genauer erläutert, weil die anderen Operationen recht einfach zu realisieren sind oder keine erwähnenswerte State-of-the-Art-Entwurfsprinzipien verwenden.

„expandAll()“

In Abbildung 6.5 ist das dynamische Verhalten der bereits erwähnten rekursiven Schablonemethode `expandAll()` in Form eines UML-Sequenzdiagramms dargestellt.

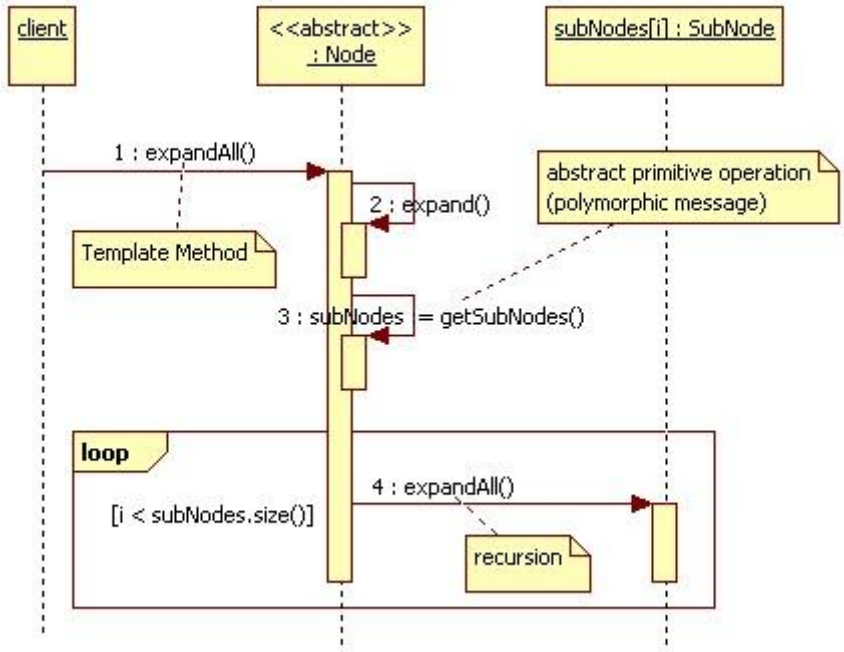

Abbildung 6.5: Systementwurf für „Knoten vollständig expandieren“

In Schritt 1 wird einem Knoten mitgeteilt, dass dieser den Knoten vollständig expandieren soll. `:Node` kann aufgrund der Vererbungsbeziehung entweder ein `CentralNode` oder ein `SubNode` sein. In Schritt 2 wird die primitive konkrete Operation `expand()` ausgeführt. Diese expandiert `:Node`. Da die gesamte Knotenhierarchie expandiert werden soll, besorgt sich `:Node` in Schritt 3 seine Unterknoten. `getSubNodes()` ist eine abstrakte primitive Operation bzw. eine polymorphe Nachricht. Die Implementierung dieser Operation stellt den variablen Teil des in der Schablonenmethode definierten Algorithmus dar. Wenn `:Node` vom Typ `SubNode` ist, werden von `getSubNodes()` die in `subNodes` enthaltenen `SubNode`s zurückgegeben. Wenn `:Node` vom Typ `CentralNode` ist, werden von `getSubNodes()` die in `leftSubNodes` und `rightSubNodes` enthaltenen `SubNode`s zurückgegeben. In Schritt 4 wird allen Unterknoten von `:Node` die Nachricht `expandAll()` zugeschickt. `expandAll()` ist damit eine rekursive Operation.

„setSuperNode(:Node)“

In Abbildung 6.6 ist der Systementwurf für die Realisierung der Operation `setSuperNode(:Node)` in Form eines UML-Sequenzdiagramms dargestellt. In Abbildung 6.7 ist auf der linken Seite die Knotenhierarchie vor Ausführung des Sequenzdia-

gramms und auf der rechten Seite die Knotenhierarchie nach Ausführung des Sequenzdiagramms dargestellt.

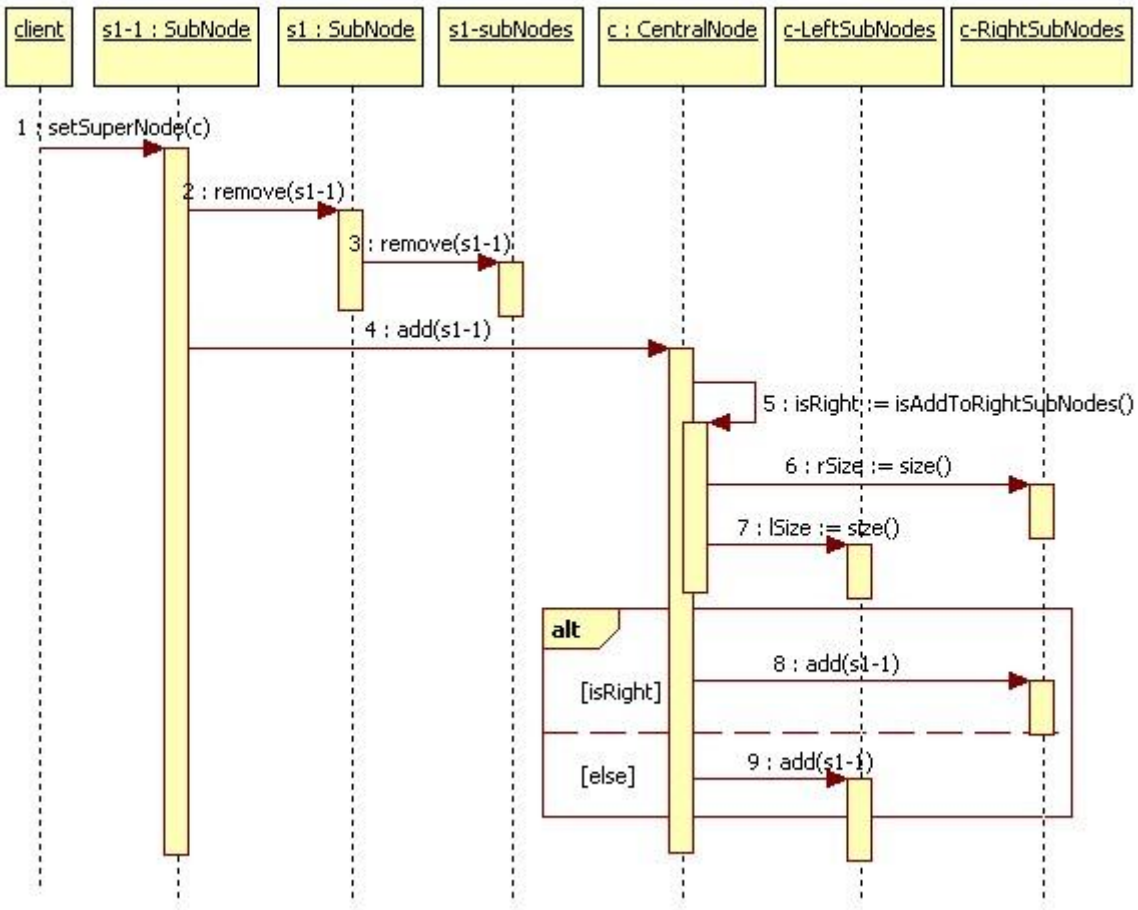

Abbildung 6.6: Systementwurf für „setSuperNode(:Node)“

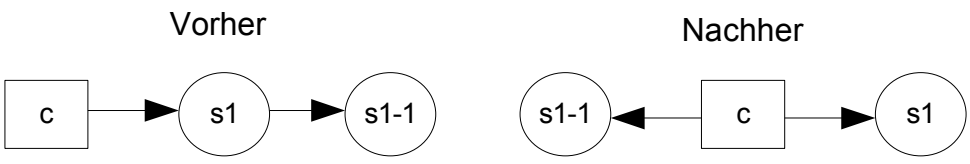

Abbildung 6.7: Knotenhierarchie vor und nach Ausführung des Sequenzdiagramms

In Schritt 1 wird dem Unterknoten `s1-1` mitgeteilt, dass der Zentralknoten `c` sein neuer Oberknoten werden soll. Da ein Knoten seine Unterknoten in einer Liste verwaltet[4], schickt `s1-1` in Schritt 2 seinem aktuellen Oberknoten `s1` die Nachricht, dass dieser ihn entfernen soll. In Schritt 4 schickt `s1-1 c` die Nachricht, dass dieser `s1-1` als neuen Unterknoten hinzufügen soll. Da es sich bei dem Knoten `c` um einen Zentralknoten (`CentralNode`) handelt, prüft dieser in Schritt 5, ob `s1-1` als rechter oder als linker Unterknoten hinzugefügt werden soll. Dazu vergleicht `c` die Anzahl seiner linken und rechten Unterknoten (Schritt 6 und 7). In der in Abbildung 6.7 dargestellten Ausgangssituation wird `s1-1` als linker Unterknoten hinzugefügt (Schritt 9), weil die Anzahl der linken Unterknoten kleiner als die Anzahl der rechten Unterknoten ist.

6.4 Schicht „gui“

Die Schicht `gui` enthält die in Abbildung 6.8 dargestellten Komponenten. Auf eine genaue Spezifikation der Operationen wird verzichtet, weil diese typischerweise Bestandteil eines GUI-Toolkits sind und daher üblicherweise nicht erst entworfen werden müssen. Es werden nur die generellen Verantwortlichkeiten der Komponenten angegeben.

[4] Ein „CentralNode“ verwaltet seine Unterknoten in „leftSubNodes“ und „rightSubNodes“. Ein „SubNode“ verwaltet seine Unterknoten in „subNodes“.

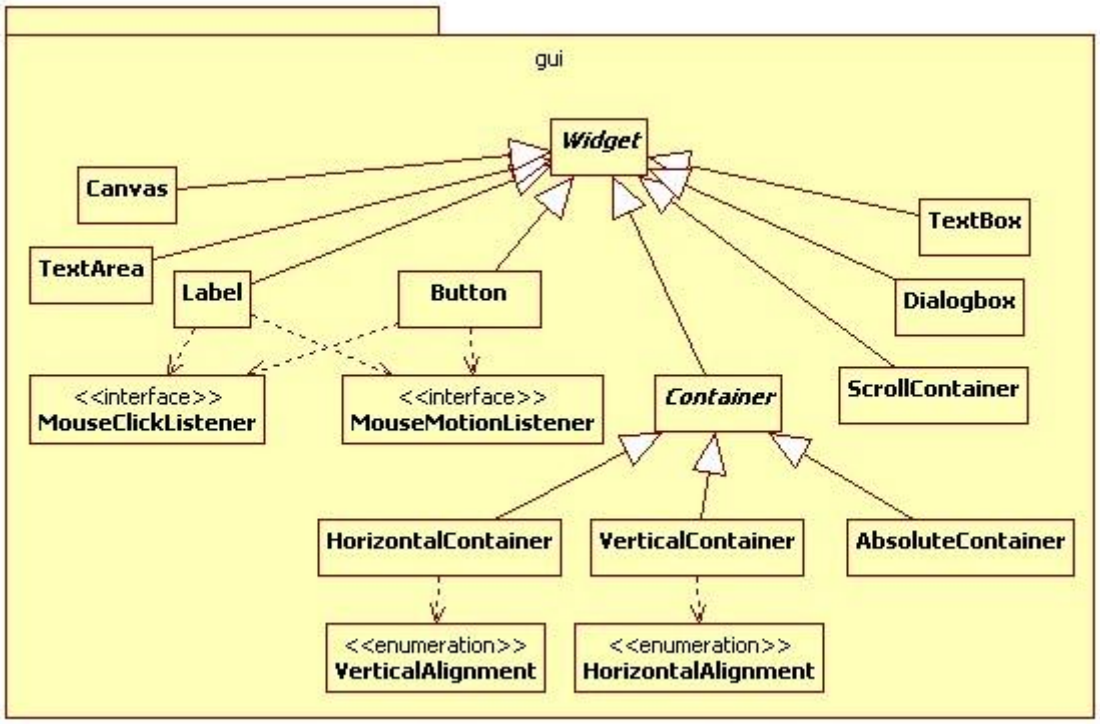

Abbildung 6.8: Schicht „gui“

Die grundlegende Klasse im Paket `gui` ist `Widget`. Ein `Widget` ist ein Element der Benutzerschnittstelle. Für `Widget`s können die folgenden Eigenschaften festgelegt werden:

- Sichtbarkeit
- Größe

Ein `Label` ist ein `Widget`, welches einen Text auf dem Bildschirm anzeigt. Der anzuzeigende Text kann vom Klienten bestimmt werden. Bei einem `Label` können `MouseClickListener` und `MouseMotionListener` registriert werden. Diese realisieren das wohlbekannte Event-Listener-Modell.

Eine `TextBox` ist ein `Widget`, das ein einzeiliges Texteingabefeld auf dem Bildschirm darstellt. Klienten von `TextBox` können den eingegebenen Text auslesen.

Eine `TextArea` ist ein `Widget`, das ein mehrzeiliges Texteingabefeld auf dem Bildschirm darstellt. Klienten von `TextArea` können den Text der `TextArea` sowohl setzen als auch auslesen.

Ein Button ist ein Widget, das eine Schaltfläche auf dem Bildschirm darstellt. Bei einem Button können MouseClickListener registriert werden. Wenn der Benutzer die Schaltfläche betätigt, werden die registrierten MouseClickListener benachrichtigt. Für einen Button kann ein Icon, das angezeigt werden soll, angegeben werden.

Ein Canvas ist ein Widget, das eine leere Fläche darstellt, auf der gezeichnet werden kann. Es stellt Operationen zum Zeichnen von Linien, Bögen, Rechtecken und anderen geometrischen Figuren zur Verfügung. Anwendungen können durch Aufrufe dieser primitiven Operationen komplexere Figuren erstellen.

Ein ScrollContainer umgibt ein Widget mit horizontalen und vertikalen Scroll-Balken.

Eine Dialogbox ist ein Fenster, das ein Widget aufnimmt, eine Überschrift besitzt und verschoben werden kann.

Ein Container ist ein Widget, das andere Widgets enthalten kann. Widgets können sowohl hinzugefügt als auch wieder entfernt werden. Da Container Widgets sind, können die enthalten Widgets selbst wieder Container sein.

HorizontalContainer, VerticalContainer und AbsoluteContainer sind Container, die ihre Elemente auf unterschiedliche Weise anordnen.

Ein HorizontalContainer ordnet die enthaltenen Widgets horizontal, d.h. in einer Zeile an. Beim einfachen Hinzufügen eines Widgets wird dieses an das rechte Ende der Zeile platziert. Alternativ kann ein Widget auch an einer bestimmten Position eingefügt werden. Für einen HorizontalContainer kann die vertikale Ausrichtung (VerticalAlignment) der enthaltenen Widgets festgelegt werden. Die vertikale Ausrichtung gibt an, ob die Elemente am oberen Rand, am unteren Rand oder im Zentrum des Containers ausgerichtet werden sollen.

Ein VerticalContainer ordnet die enthaltenen Widgets vertikal, d.h. in einer Spalte an. Beim einfachen Hinzufügen eines Widgets wird dieses an das untere Ende der Spalte platziert. Alternativ kann ein Widget auch an einer bestimmten Position eingefügt werden. Für einen VerticalContainer kann die horizontale Ausrichtung (HorizontalAlignment) der enthaltenen Widgets festgelegt werden. Die horizontale Ausrichtung gibt an, ob die Elemente am rechten Rand, am linken Rand oder in der Mitte des Containers ausgerichtet werden sollen.

Ein AbsoluteContainer ordnet die enthaltenen Widgets nach absoluten Positionsangaben an. Beim Hinzufügen der Widgets müssen für diese die x- und die y-Koordinate angegeben werden.

6.5 Schicht „presentation“

In diesem Abschnitt wird der Systementwurf für die Schicht `presentation` beschrieben.

Im Unterabschnitt 6.5.1 wird die Paketstruktur der Schicht `presentation` beschrieben. Die einzelnen Pakete werden in den darauffolgenden Unterabschnitten beschrieben.

Die Kopplung zwischen der Schicht `presentation` und der Schicht `domain` wird im Abschnitt 6.6 beschrieben. Die Kopplung wird nicht in diesem Abschnitt beschrieben, damit die UML-Diagramme für die Schicht `presentation` übersichtlich bleiben.

6.5.1 Pakete

In Abbildung 6.9 wird die Paketstruktur der Schicht `presentation` dargestellt.

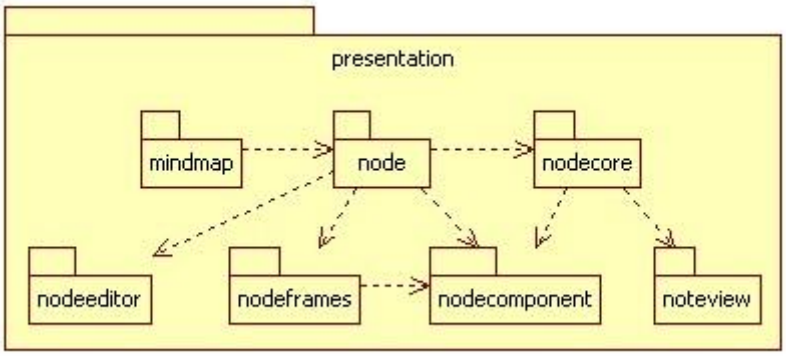

Abbildung 6.9: Paketstruktur der Schicht „presentation“

Das Ziel bei der Definition der Paketstruktur ist, die Auswirkung von Änderungen an der Software möglichst lokal zu halten [Larman (2005)]. In [Larman (2005)] werden die folgenden Richtlinien zur Definition der Paketstruktur beschrieben:

- Der Zusammenhalt eines Pakets sollte möglichst hoch sein.
- Die Kopplung zwischen Paketen sollte möglichst niedrig sein.
- Die Pakete sollten keine zyklischen Abhängigkeiten aufweisen.
- Typen, die unabhängig voneinander verwendet werden können, sollten auch in verschiedenen Paketen angesiedelt sein.

Die in Abbildung 6.9 dargestellte Paketstruktur besitzt keine zyklischen Abhängigkeiten[5] zwischen den Paketen. Die Abhängigkeiten zwischen den Paketen sind recht niedrig. Im Folgenden werden die Aufgaben der einzelnen Pakete beschrieben.

Das Paket `noteview` realisiert die Notiz-Anzeigedialogbox[6].

Das Paket `nodeeditor` realisiert die Link-Bearbeitungsdialogbox[7] und die Notiz-Bearbeitungsdialogbox[8]. Die Link- und die Notiz-Bearbeitungsdialogbox werden im gleichen Paket realisiert, weil diese sich nur wenig voneinander unterscheiden. Diese unterscheiden sich lediglich durch den Dialogbox-Titel, den initialen Text in der Textbox und der Aktion beim Drücken der „Accept“-Schaltfläche.

Das Paket `nodecomponent` realisiert ein kleines Framework zur Unterstützung auswechselbarer Knotenrahmen.

Das Paket `nodeframes` erweitert das Framework aus dem Paket `nodecomponent` um konkrete Knotenrahmen, welche die Kreisrahmen, Rechteckrahmen und Unterlinierahmen realisieren.

Das Paket `nodecore` erweitert das Framework aus dem Paket `nodecomponent` und realisiert den Knotenkern.

Die Pakete `nodeframes` und `nodecore` realisieren den äußeren und den inneren Teil eines Knotens. Das Paket `node` kombiniert diese beiden zu einem vollwertigen Knoten. Aus Sicht von Klienten erscheint ein Knoten atomar.

Das Paket `mindmap` ist für die Anordnung der Knoten zuständig. In diesem Paket wird z.B. der Mechanismus für die radiale Mind-Map-Struktur realisiert. Dieses Paket ist auch für die Erstellung der Kanten zwischen den Knoten zuständig.

Die Pakete besitzen klar abgegrenzte Aufgaben und weisen damit einen hohen Zusammenhalt auf.

6.5.2 Paket „noteview“

In Abbildung 6.10 ist der Systementwurf für das Paket `noteview` dargestellt.

[5] Dies erkennt man daran, dass die Abhängigkeitspfeile alle in eine Richtung zeigen.
[6] siehe Abbildung 5.5
[7] siehe Abbildung 5.3
[8] siehe Abbildung 5.4

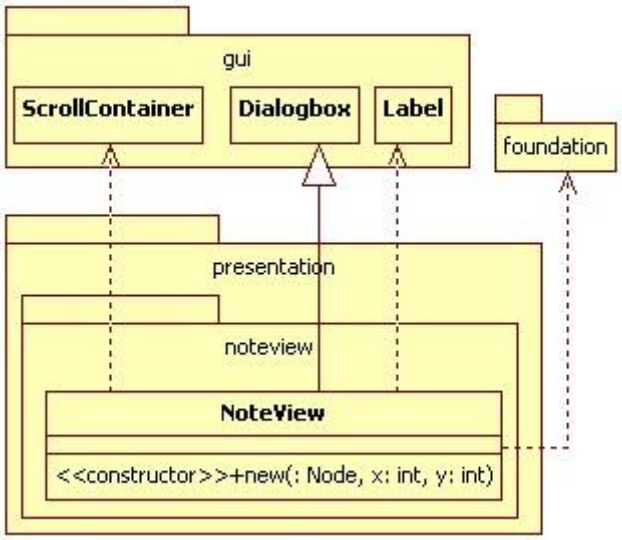

Abbildung 6.10: Paket „noteview“

`NoteView` realisiert die Notiz-Anzeigedialogbox[9]. Die Knotennotiz wird von einem `Label` angezeigt. Da Notizen beliebig lang sein können, wird das `Label` von einem `ScrollContainer` umgeben. Beim Erzeugen einer `NoteView` wird die Position, an der die Dialogbox angezeigt werden soll, mitgeteilt.

6.5.3 Paket „nodeeditor“

In Abbildung 6.11 ist der Systementwurf für das Paket `nodeeditor` dargestellt.

[9] siehe Abbildung 5.5

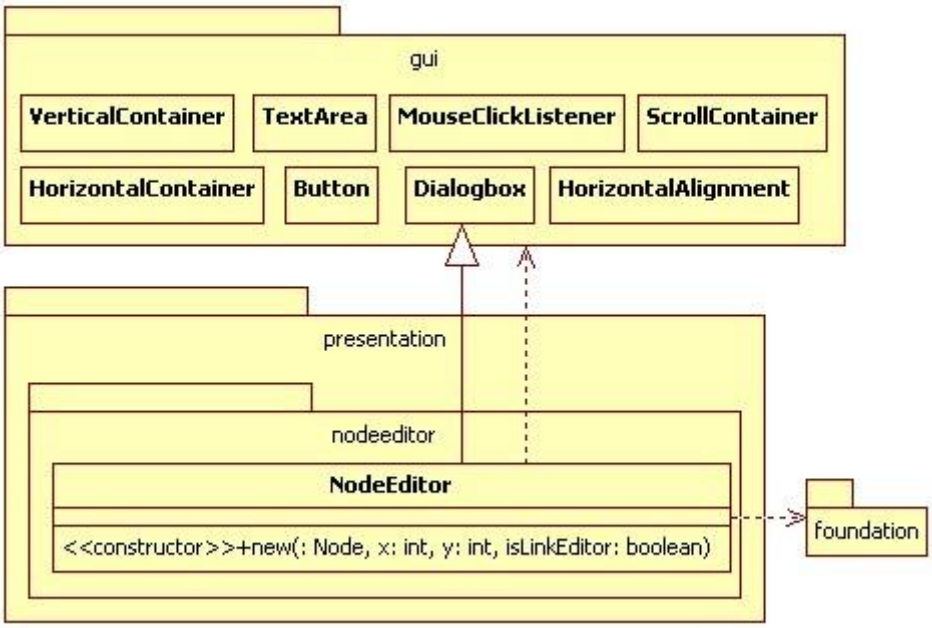

Abbildung 6.11: Paket „nodeeditor“

`NodeEditor` realisiert sowohl die Notiz-Bearbeitungsdialogbox[10] als auch die Link-Bearbeitungsdialogbox[11]. Die Notiz bzw. der Link wird in eine `TextArea` eingegeben. Da eine Notiz bzw. ein Link beliebig lang sein kann, wird die `TextArea` vom einem `ScrollContainer` umgeben. Für die Schaltflächen „Accept“ und „Cancel“ wird jeweils ein `Button` verwendet. Diese `Button`s werden in einen `HorizontalContainer` platziert. Die `TextArea` und der `Button`-Container werden in einen `VerticalContainer` platziert. Für den `VerticalContainer` wird das `HorizontalAlignment` auf rechts gesetzt, so dass der `Button`-Container am rechten Rand der Dialogbox erscheint. Für die `Button`s werden `MouseClickListener` registriert, so dass beim Betätigen der `Button`s die entsprechenden Aktionen ausgeführt werden. Beim Erzeugen eines `NodeEditor`s wird die Position, an der die Dialogbox angezeigt werden soll und, ob eine Link-Bearbeitungsdialogbox oder eine Notiz-Bearbeitungsdialogbox erstellt werden soll, mitgeteilt.

[10] siehe Abbildung 5.4
[11] siehe Abbildung 5.3

6.5.4 Paket „nodecomponent“

In den Anforderungen[12] wurde beschrieben, dass ein Benutzer den Rahmen eines Knotens festlegen kann. Beim Wechsel des Knotenrahmens ändert sich an der Darstellung des Knotenkerns nichts. In diesem Abschnitt wird ein kleines Framework zur Unterstützung auswechselbarer Knotenrahmen beschrieben.

Vererbung

In Abbildung 6.12 wird ein auf Vererbung basierender Lösungsansatz zur Unterstützung auswechselbarer Knotenrahmen dargestellt.

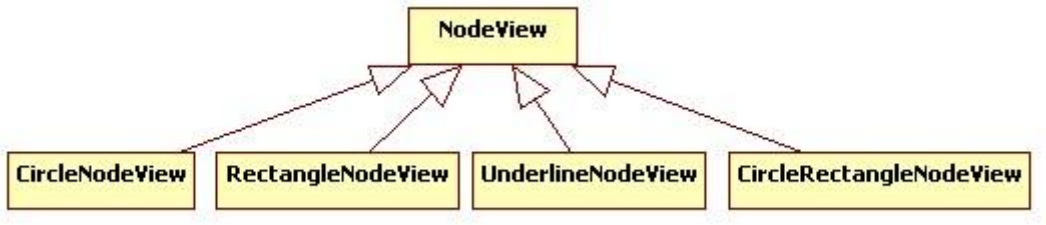

Abbildung 6.12: Auswechselbare Knotenrahmen mit Vererbung

Ein `NodeView` ist für die Darstellung des Knotenkerns zuständig. Die Unterklassen von `NodeView` erweitern die Darstellung `NodeView` um einen Knotenrahmen. Es gibt für die unterschiedlichen Knotenrahmen (z.B. Kreisrahmen, Rechteckrahmen) jeweils eine Unterklasse.

Bei diesem Lösungsansatz muss jedesmal, wenn der Knotenrahmen gewechselt wird, der komplette Knotenkern neu erstellt werden. Diese Erstellung ist unnötig, da sich am Knotenkern selbst nichts ändert.

Ein weiterer Nachteil dieses Lösungsansatzes ist, dass Knotenrahmen nicht einfach miteinander kombiniert werden können. Z.B. wäre es denkbar, dass MiMa in Zukunft einen Knotenrahmen, der eine Kombination aus Rechteckrahmen und Kreisrahmen darstellt (`CircleRectangleNodeView`), unterstützen soll. Bei diesem Lösungsansatz könnten die vorhandenen Klassen `CircleNodeView` und `RectangleNodeView` für den neuen `CircleRectangleNodeView` nicht wiederverwendet werden. Es müsste eine neue Unterklasse `CircleRectangleNodeView` von `NodeView` erstellt werden, die sowohl

[12] siehe Unterabschnitt 5.3.8

Code zur Darstellung eines Kreises als auch Code zur Darstellung eines Rechtecks beinhaltet.

Entwurfsmuster Dekorierer

In Abbildung 6.13 ist ein auf dem Entwurfsmuster Dekorierer basierender Lösungsansatz zur Unterstützung auswechselbarer Knotenrahmen dargestellt. Das Entwurfsmuster Dekorierer ist in Abschnitt 4.1.3 beschrieben.

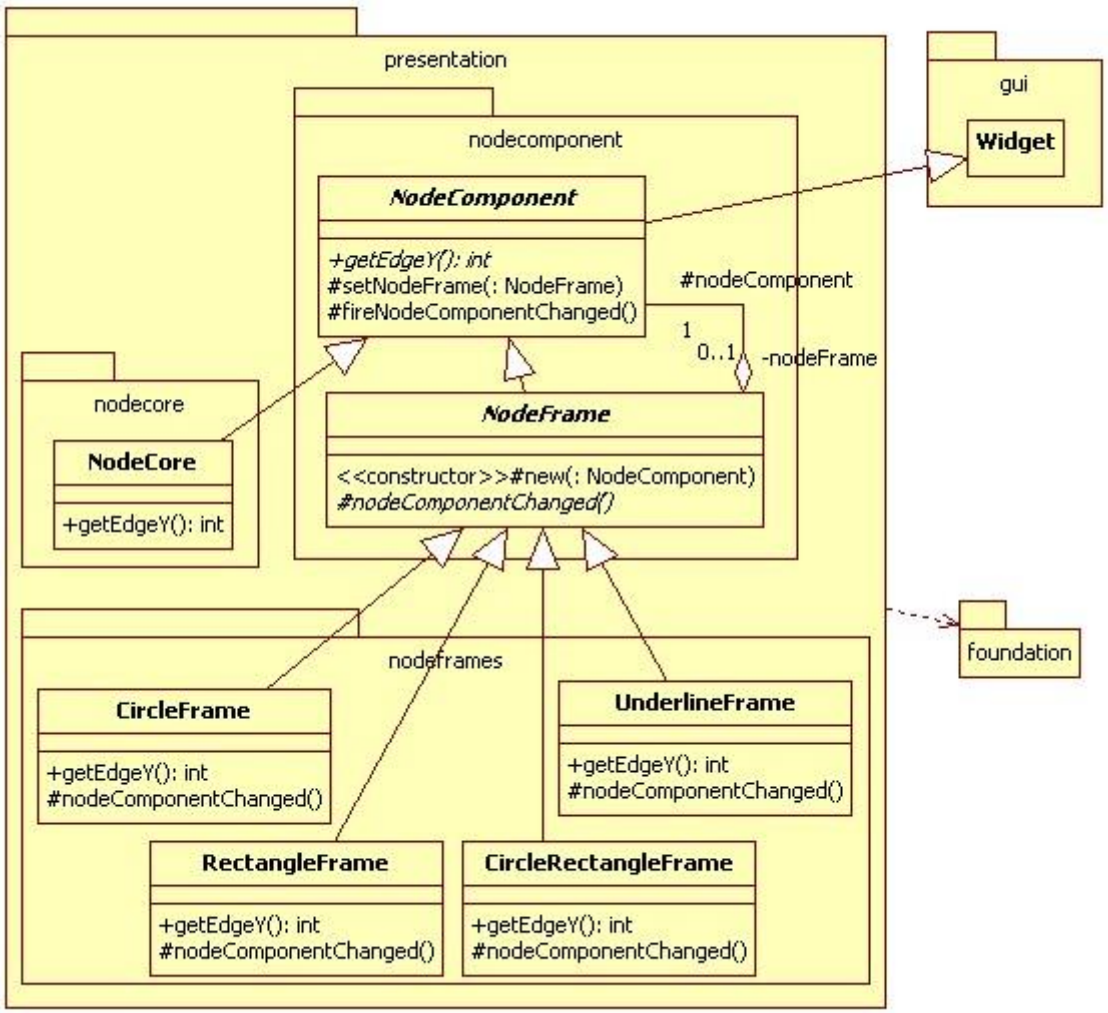

Abbildung 6.13: Auswechselbarer Knotenrahmen mit Entwurfsmuster Dekorierer

In Tabelle 6.2 werden die Klassen aus Abbildung 6.13, den Klassen des Dekorierer-Musters zugeordnet.

Knotenrahmen	**Entwurfsmuster Dekorierer**
`NodeComponent`	`Komponente`
`NodeCore`	`KonkreteKomponente`
`NodeFrame`	`Dekorierer`
`CircleFrame`, `RectangleFrame` und `UnderlineFrame`	`KonkreterDekorierer`

Tabelle 6.2: Abbildung Knotenrahmen auf Dekorierer

Ein `NodeCore` ist für die Darstellung des Knotenkerns zuständig. Ein `NodeFrame` bzw. genauer die Unterklassen von `NodeFrame` sind für die Darstellung des Knotenrahmens zuständig.

In Abschnitt 5.4.1 wurde die Anforderung, dass ein Unterknoten immer über eine Kante mit seinem Oberknoten verbunden ist, beschrieben. Je nachdem welchen Knotenrahmen ein Knoten besitzt, ist der Verbindungspunkt einer Kante zum Knoten unterschiedlich. Z.B. ist bei einem Kreisrahmen der Verbindungspunkt zentral am linken Rand des Knotens und bei einem Unterlinierahmen am unteren linken Rand des Knotens. In der Klasse `NodeComponent` ist die abstrakte Operation `getEdgeY():int` zur Erfragung des Verbindungspunkts definiert. Diese Operation wird von den Unterklassen von `NodeFrame` implementiert.

Das Dekorierer-Muster kann zur Unterstützung auswechselbarer Knotenrahmen nicht 1:1 übernommen werden. Im Gegensatz zum reinen Dekorierer-Muster besitzt die Komponente (hier `NodeComponent`) noch eine Referenz auf seinen Dekorierer (hier `NodeFrame`). Dies ist notwendig, da ein `NodeCore` seinen umgebenden `NodeFrame` benachrichtigen muss, wenn sich die Größe von `NodeCore` ändert. Die Größe von `NodeCore` ändert sich z.B. wenn der Knotentitel verändert wird. Wenn sich die Größe von `NodeCore` ändert, muss auch die Größe des Knotenrahmens angepasst werden. Die geschützte Operation `setNodeFrame(:NodeFrame)` wird von einem `NodeFrame` bei dessen Erzeugung für die dekorierte Komponente (z.B. `NodeCore`) aufgerufen. Wenn `NodeCore` bzw. allgemeiner eine `NodeComponent` sich in der Darstellung verändert, so dass ein Neuzeichnen des umgebenden Knotenrahmens erforderlich ist, ruft `NodeCore` bzw. allgemeiner die `NodeComponent` die geschützte Operation `fireNodeComponentChanged()` auf.

Zur Unterstützung auswechselbarer Knotenrahmen wird in dieser Arbeit der Dekorierer-Ansatz realisiert, weil dieser die Nachteile des auf Vererbung basierenden Ansatzes aufhebt.

6.5.5 Paket „nodecore“

In Abbildung 6.14 ist der Systementwurf für das Paket `nodecore` dargestellt.

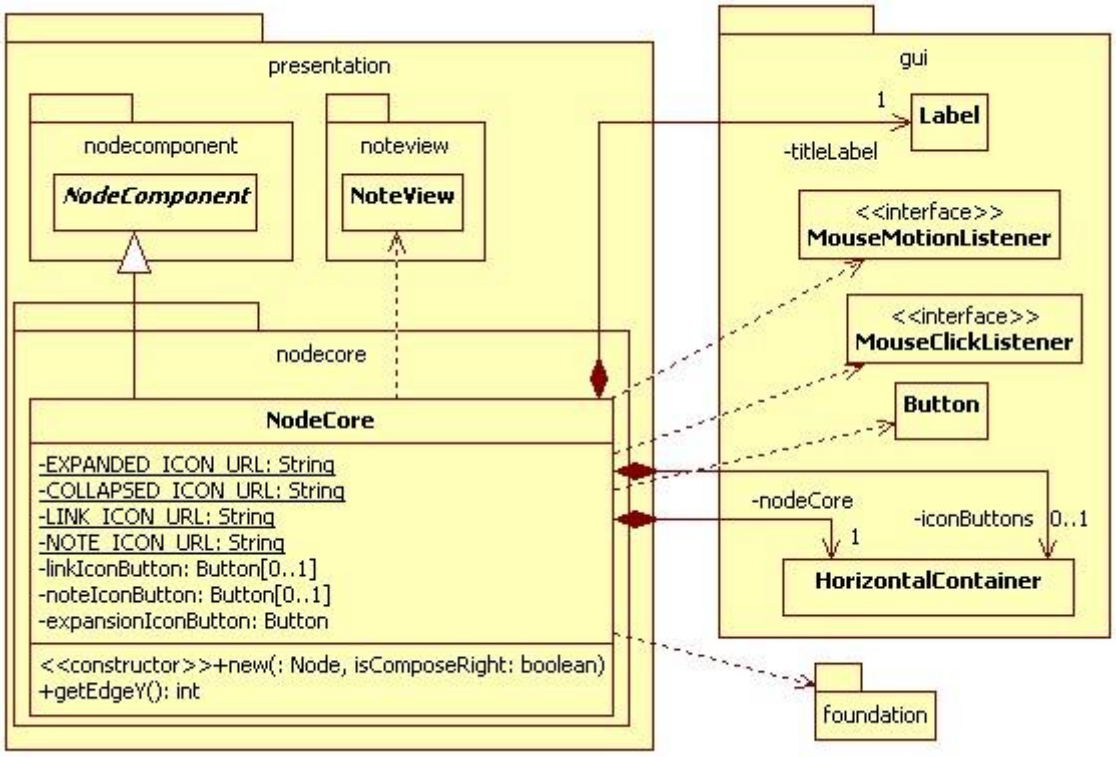

Abbildung 6.14: Paket „nodecore“

`NodeCore` realisiert den Knotenkern. `titleLabel` ist für die Anzeige des Knotentitels zuständig. `NodeCore` registriert sich bei `titleLabel` als `MouseMotionListener`. Wenn der Mauszeiger den Bildschirmbereich von `titleLabel` betritt oder verlässt, wird `NodeCore` von `titleLabel` benachrichtigt. Damit kann `NodeCore` die Hintergrundfarbe von `titleLabel` aktualisieren.

`NodeCore` besitzt `Button`s für das Link-Icon, das Note-Icon und das Expansion-Icon. Für diese wird die URL des anzuzeigenden Icons angegeben. `NodeCore` registriert sich bei den `Button`s aus dem gleichen Grund wie bei `titleLabel` als `MouseMotionListener`. `linkIconButton` und `noteIconButton` werden in den `HorizontalContainer` `iconButtons` eingefügt.

Ein Knotenkern ordnet seine Elemente (Knotentitel, Icons) horizontal an. `iconButtons`, `titleLabel` und `expansionIconButton` werden daher in den `HorizontalContainer` `nodeCore` eingefügt.

In Abbildung 6.15 wird die Struktur von `NodeCore` veranschaulicht. Die Bezeichnungen entsprechen den Eigenschaften der Klassen in Abbildung 6.14.

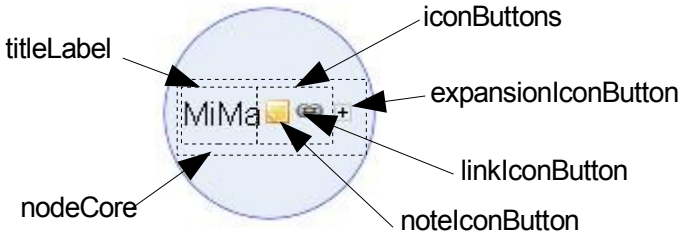

Abbildung 6.15: Beispiel Knotenkern

6.5.6 Paket „nodeframes“

In Abbildung 6.11 ist der Systementwurf für das Paket `nodeframes` dargestellt.

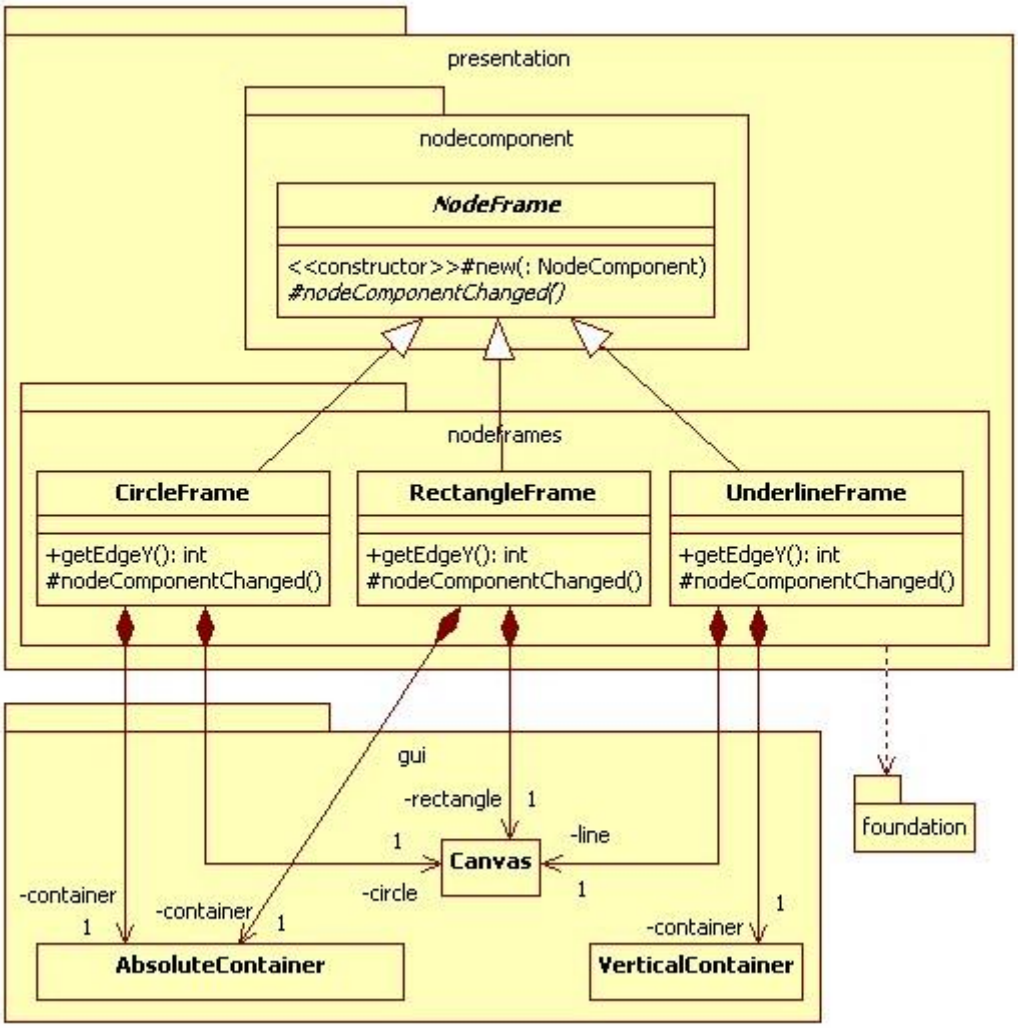

Abbildung 6.16: Paket „nodeframes“

Die Beziehungen der Klassen `CircleFrame`, `RectangleFrame` und `UnderlineFrame` zur Klasse `NodeFrame` sind in Abschnitt 6.5.4 beschrieben und werden daher an dieser Stelle nicht weiter erläutert.

`CircleFrame`, `RectangleFrame` und `UnderlineFrame` besitzen jeweils ein `Canvas`, in das sie den Knotenrahmen zeichnen. `CirlceFrame` und `RectangleFrame` fügen die dekorierte `nodecomponent` und ihr `Canvas` in einen `AbsoluteContainer` ein, um die dekorierten `nodecomponent`s im Zentrum ihrer Knotenrahmen zu platzieren. `UnderlineFrame` platziert ihre `nodecomponent`

und ihr `Canvas` in einen `VerticalContainer`, um ihr `Canvas` unterhalb der `nodecomponent` zu platzieren.

6.5.7 Paket „node“

In Abbildung 6.17 ist der Systementwurf für das Paket `node` dargestellt.

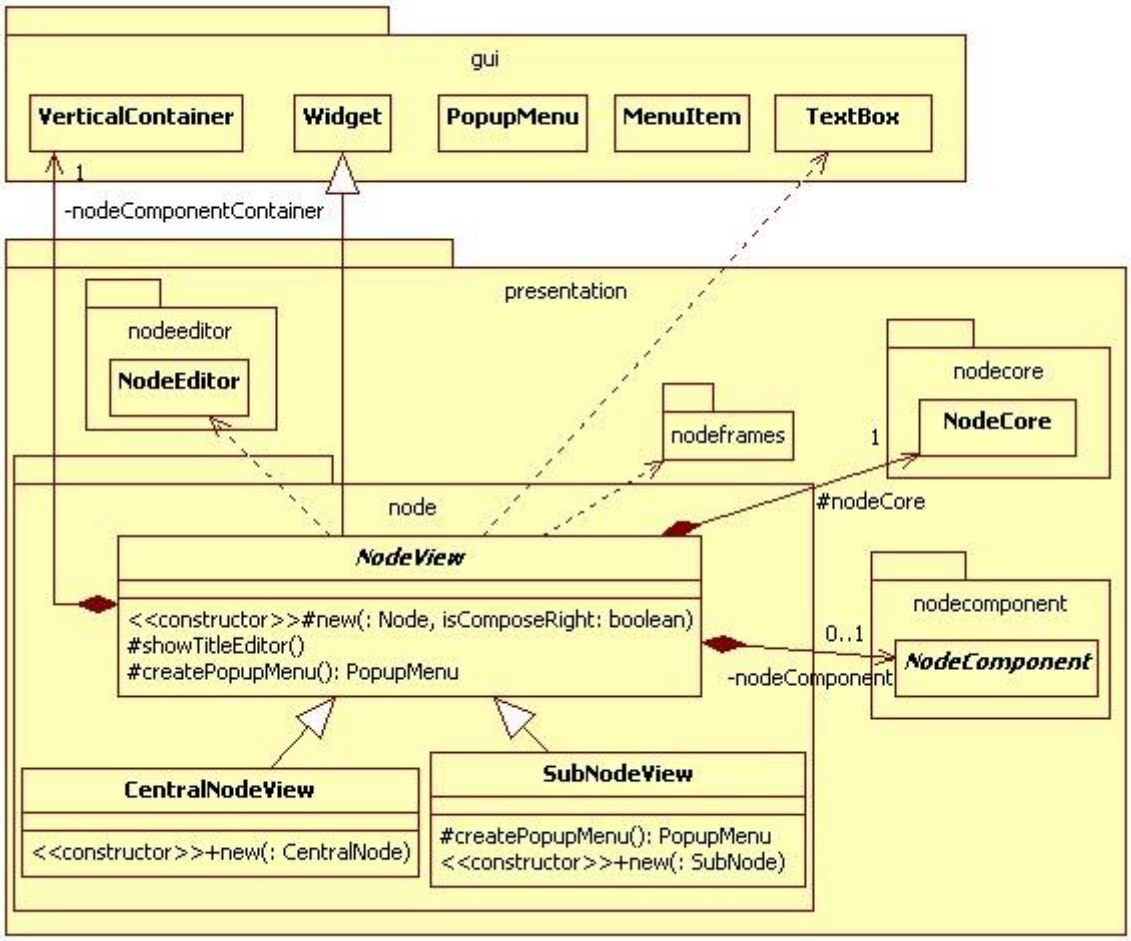

Abbildung 6.17: Paket „node“

Die Verantwortlichkeiten von `NodeView` sind:

- Wechsel des Knotenrahmens

- Zusammenbau und Anzeige des Popup-Menus, wenn mit der rechten Maustaste auf den Knoten (`NodeView`) geklickt wird
- Anzeigen eines Texteingabefelds und Verbergen des Knotenkerns und Knotenrahmens, wenn der Knotentitel bearbeitet werden soll

Die Verantwortlichkeiten von `SubNodeView` und `CentralNodeView` sind, das Popup-Menu um Unterknoten- bzw. Zentralknoten-spezifische Menu-Einträge zu erweitern.

6.5.8 Paket „mindmap“

In Abbildung 6.18 ist der Systementwurf für das Paket `mindmap` dargestellt.

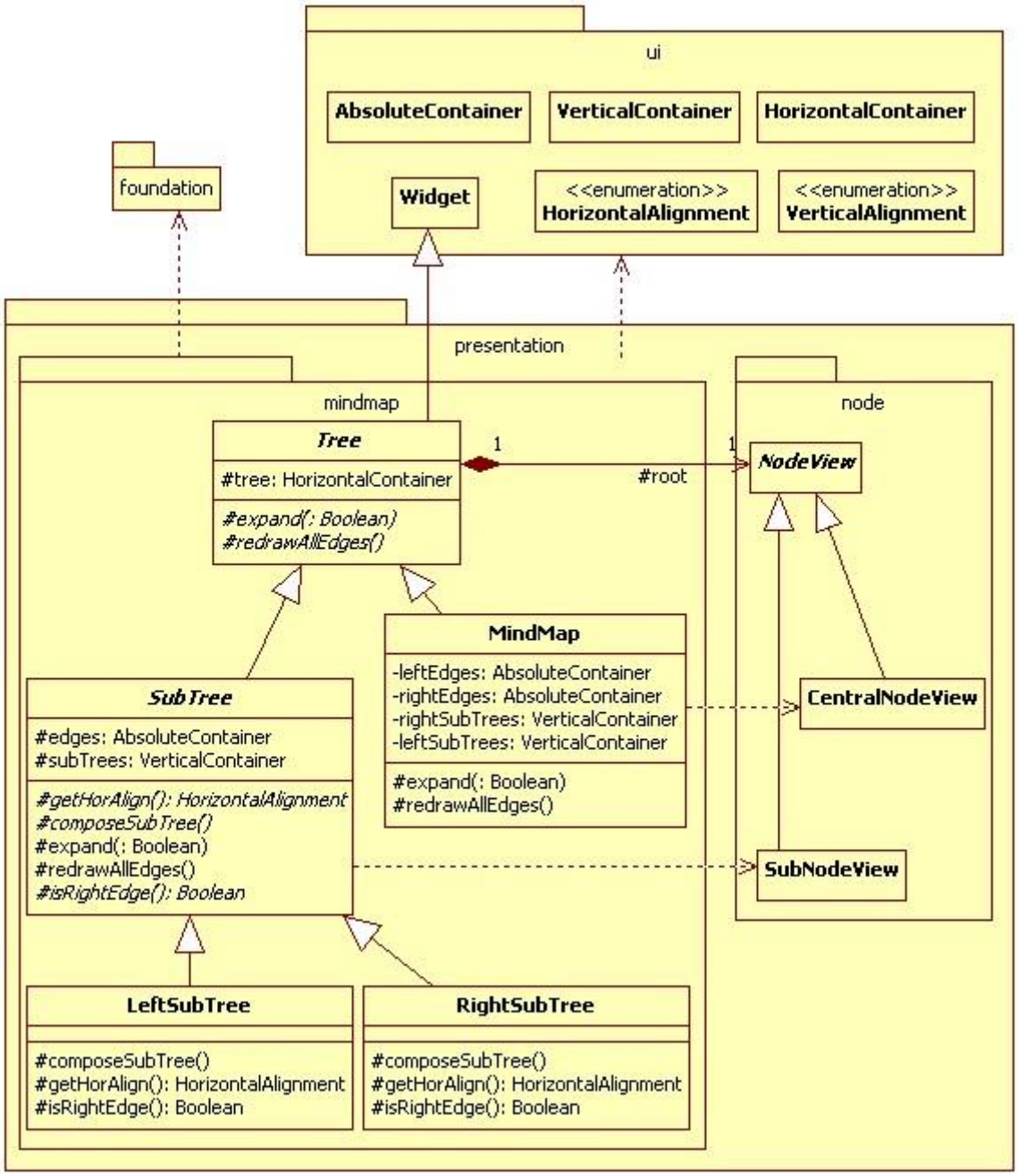

Abbildung 6.18: Paket „mindmap“

Das Paket `mindmap` realisiert die radiale Mind-Map-Struktur[13]. Die Grundlage für die radiale Mind-Map-Struktur bilden `HorizontalContainer` und `VerticalContainer`. Die radiale Mind-Map-Struktur wird durch Verschachtelungen von `HorizontalContainer` und `VerticalContainer` realisiert.

Ein `Tree` stellt einen Teilbaum der Mind-Map dar. Ein Teilbaum besteht aus einer Wurzel (`root`), seinen Unterbäumen (`SubTree`s) und Kanten zwischen der Wurzel und seinen Unterbäumen. Ein `Tree` ist im Wesentlichen ein `HorizontalContainer`. Ein `Tree` nimmt `root`, seine Unterbäume und die Kanten in einem `HorizontalContainer` auf. `Tree` wird nicht von `HorizontalContainer` abgeleitet, weil dann die Klienten von `Tree` auf den `HorizontalContainer` direkt zugreifen könnten und dadurch die Kapselung von `Tree` aufgebrochen wäre. Stattdessen besitzt `Tree` die gleichnamige Instanzvariable `tree`. Das `VerticalAlignment` von `tree` wird auf mittig gesetzt.

Ein Teilbaum kann entweder die gesamte Mind-Map (`MindMap`) oder ein Unterbaum (`SubTree`) sein. Eine `MindMap` besitzt linke Unterbäume (`LeftSubTree`s) und rechte Unterbäume (`RightSubTree`s). Die linken Unterbäume werden in den `VerticalContainer leftSubTrees` und die rechten Unterbäume in den `VerticalContainer rightSubTrees` platziert. Die Kanten zu den linken Unterbäumen werden in den `AbsoluteContainer leftEdges` platziert. Die Kanten zu den rechten Unterbäumen werden in den `AbsoluteContainer rightEdges` platziert. Die Kanten können nicht genauso wie die Unterbäume in einen `VerticalContainer` platziert werden, weil sich die Widgets für die Kanten zur korrekten Darstellung überlagern müssen. Eine `MindMap` fügt `leftSubTrees`, `leftEdges`, `root`, `rightEdges` und `rightSubTrees` in dieser Reihenfolge zum Container `tree` hinzu.

Eine `SubTree` ist einer `MindMap` vom Aufbau ähnlich, besitzt aber nur linke oder nur rechte Unterbäume. Die Klasse `SubTree` besitzt die Unterklassen `LeftSubTree` und `RightSubTree`, weil sich die Anordnung der Komponenten des Containers `tree` für linke und rechte Unterbäume unterscheidet.

In Abbildung 6.19 ist ein Beispiel für die in Abbildung 6.18 gezeigte Struktur dargestellt. Die Bezeichnungen in Abbildung 6.19 entsprechen dabei den Bezeichnungen in Abbildung 6.18.

[13] siehe Unterabschnitt 5.4.2

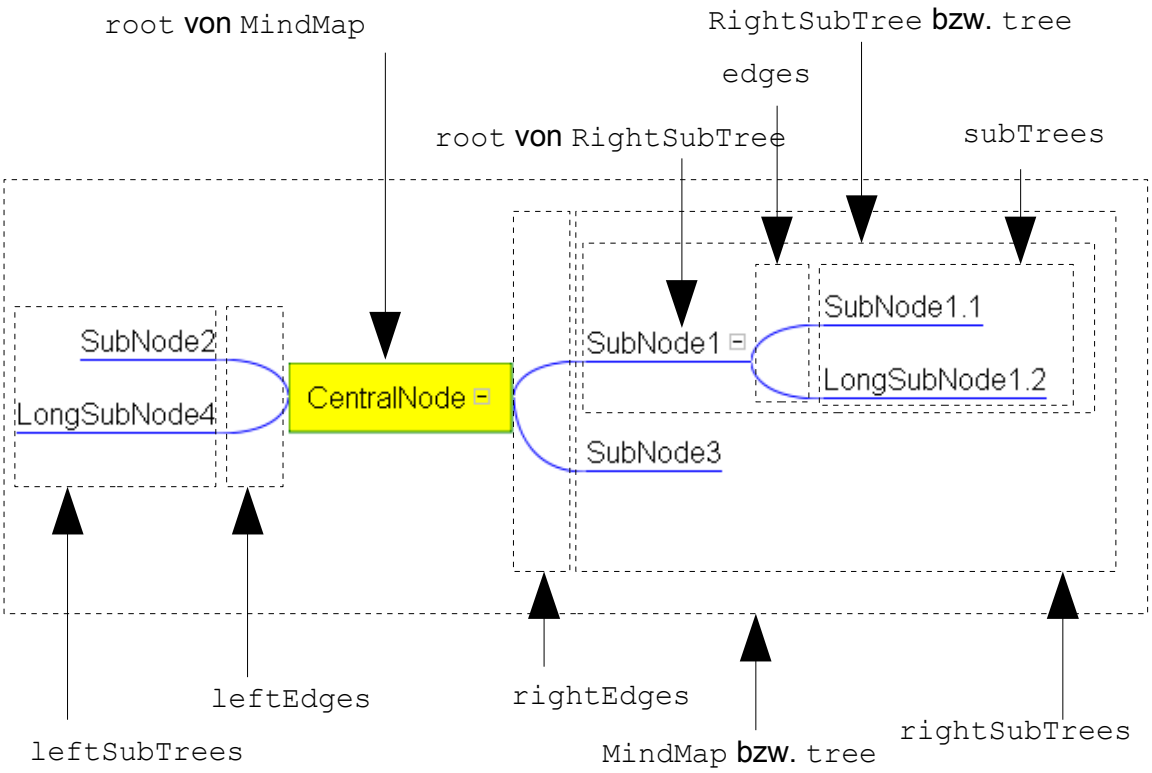

Abbildung 6.19: Beispiel radiale Mind-Map-Struktur

6.6 Kopplung der Schichten „presentation“ und „domain“

Die Schichten `presentation` und `domain` wurden in den Abschnitten 6.5 und 6.3 voneinander isoliert betrachtet. In diesem Abschnitt wird nun die Kopplung, d.h. das Zusammenspiel, zwischen den Schichten `presentation` und `domain` beschrieben.

Auf der einen Seite gibt es in der Schicht `presentation` Klassen, die den Domänenzustand verändern und auf der anderen Seite gibt es Klassen, die den Domänenzustand abfragen. Es gibt auch Klassen, die den Domänenzustand sowohl verändern als auch abfragen. Abbildung 6.20 zeigt ein Beispiel.

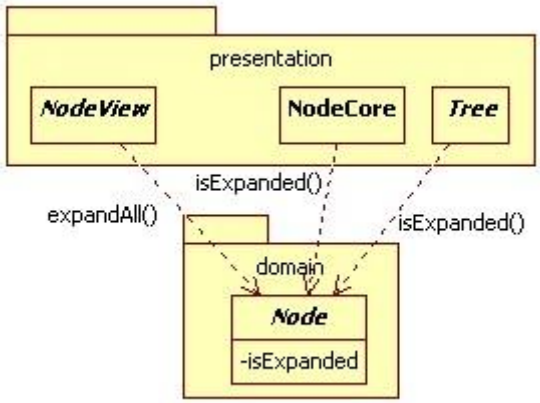

Abbildung 6.20: Verändern und Abfragen des Domänenzustands

Die Klasse `NodeView` verändert den Domänenzustand, weil diese die Operation `expandAll()` von `Node` aufruft, wenn ein Benutzer den Menu-Eintrag „Expand All Subnodes“ anklickt. `expandAll()` verändert den Zustand von `isExpanded`. Sowohl die Klasse `NodeCore` als auch die Klasse `Tree` sind vom Domänenzustand `isExpanded` abhängig. `NodeCore` ruft die Operation `isExpanded()` auf, um das Expansion-Icon zu aktualisieren. `Tree` ruft `isExpanded()` auf, um seine Unterbäume zu verbergen bzw. anzuzeigen. Es ist erforderlich, dass die Schichten ihre Zustände synchronisieren. Z.B. sollten `NodeCore` und `Tree` benachrichtigt werden, wenn der Domänenzustand `isExpanded` verändert wird.

Die Synchronisierung der Schichten `presentation` und `domain` wird auf Grundlage des Entwurfsmusters Beobachter realisiert. Das Beobachter-Muster ist in Abschnitt 4.1.2 beschrieben. In Abbildung 6.21 ist die Anwendung des Beobachter-Musters dargestellt.

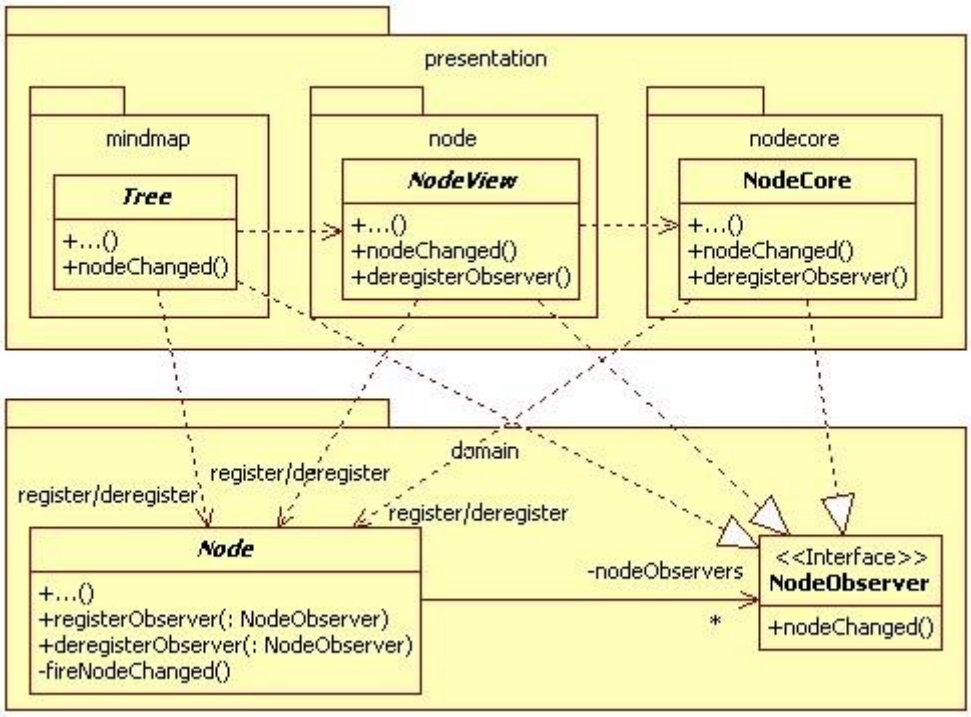

Abbildung 6.21: Entwurfsmuster Beobachter

Die Klasse `Node` ist das Subjekt. `Node` verwaltet eine Liste von `NodeObserver`-Objekten. Die Klassen `Tree`, `NodeView` und `NodeCore` sind die Beobachter von `Node` und implementieren daher die Schnittstelle `NodeObserver`. Diese registrieren sich bei `Node`, um bei Zustandsänderungen von `Node` benachrichtigt zu werden. Alle Operationen von `Node`, die den Zustand von `Node` ändern, rufen nach der Zustandsänderung die Operation `fireNodeChanged()` auf. Diese schickt allen registrierten `NodeObserver`n die Nachricht `nodeChanged()`. Wenn ein `NodeObserver` die Nachricht `nodeChanged()` empfängt, fragt dieser den Zustand von `Node` ab, um die Darstellung zu aktualisieren.

6.7 Fazit

In diesem Kapitel wurde für die Anforderungen aus Kapitel 5 ein Systementwurf vorgestellt. Im Systementwurf wurden die folgenden State-of-the-Art-Entwurfsprinzipien verwendet:

- Architekturmuster

- Entwurfsmuster
- Modularisierung

Als Architekturmuster wurden Separated-Presentation und Layers verwendet. Als Entwurfsmuster wurden Schablonenmethode, Dekorierer und Beobachter verwendet. Der beschriebene Systementwurf ist insgesamt sehr modular aufgebaut.

Um die Frage

> *Können mit dem Google Web Toolkit 2D-Ajax-Anwendungen nach den State-of-the-Art-Entwurfsprinzipien gebaut werden?*

aus dem Einleitungskapitel zu beantworten, sollte als nächstes der Systementwurf aus diesem Kapitel mit dem GWT realisiert werden.

7 Implementierung und Evaluation

In diesem Kapitel wird beschrieben, wie gut der Systementwurf aus Kapitel 6 mit dem GWT implementiert werden konnte. Als Entwicklungsumgebung wurde Eclipse eingesetzt. Das gesamte Eclipse-Projekt inklusive Quellcode, verwendeter Bibliotheken, Javadoc, Unit-Tests und der ausführbaren Anwendung sind der CD in Anhang A beigelegt.

7.1 Schichtenarchitektur

Die in Abschnitt 6.1 beschriebene Schichtenarchitektur konnte mit dem GWT implementiert werden. In Abbildung 7.1 ist die Abbildung der Schichtenarchitektur auf Java-Pakete dargestellt.

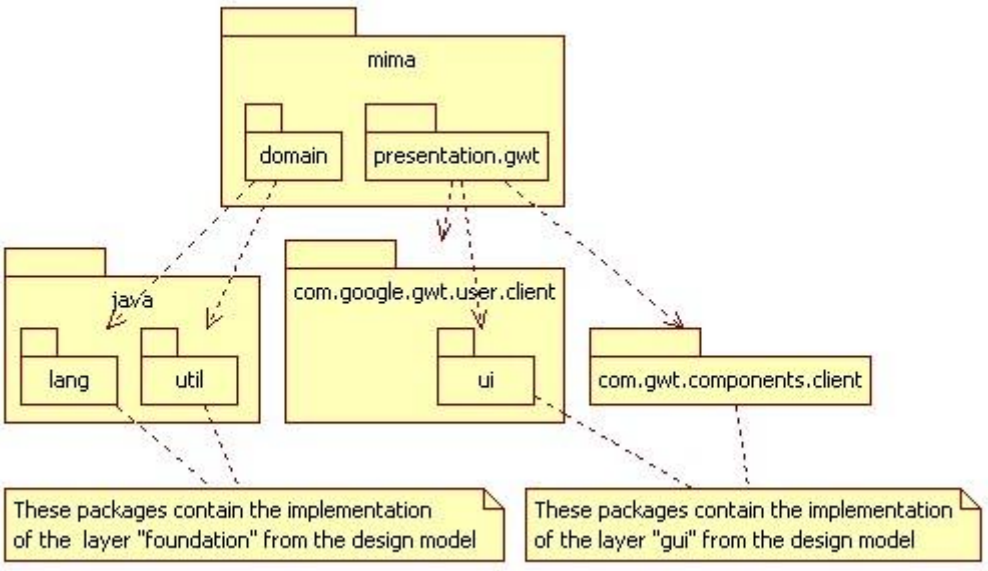

Abbildung 7.1: Implementierung der Schichtenarchitektur

Die Schicht `foundation` steht im GWT in Form der emulierten Java-Pakete `java.lang` und `java.util` zur Verfügung und musste daher nicht selbst implementiert werden. Obwohl die emulierten Java-Pakete nicht alle Klassen und Methoden der „echten“ Java-Pakete `java.lang` und `java.util` unterstützten, gibt es für alle im Systementwurf definierten Klassen eine entsprechende Implementierung.

Die Implementierung der Schicht `domain` wird im Paket `mima.domain` zusammengefasst. Die Implementierung wird im Abschnitt 7.2 beschrieben.

Die Schicht `gui` steht im GWT in Form der Java-Pakete `com.google.gwt.user.client` und `com.google.gwt.user.client.ui` zur Verfügung und musste daher nicht selbst implementiert werden. Im Wesentlichen enthalten die Pakete für alle im Systementwurf definierten Klassen der Schicht `gui` entsprechende Implementierungen. Nur für die Klasse `Canvas` gibt es im GWT keine entsprechende Implementierung. Die Klasse `Canvas` musste dennoch nicht selbst implementiert werden, weil die Implementierung von Alexei Sokolov verwendet werden konnte [Sokolov (o.J.)]. Diese wird im Paket `com.gwt.components.client` bereitgestellt. Da es noch andere Möglichkeiten für die Implementierung einer `Canvas`-Klasse gibt, wird im Abschnitt 7.3 dargestellt, warum gerade die Implementierung von Alexei Sokolov verwendet wurde.

Die Implementierung der Schicht `presentation` wird im Paket `mima.presentation.gwt` zusammengefasst. Diese wird nicht direkt im Paket `mima.presentation` zusammengefasst, weil in Zukunft möglicherweise weitere Implementierungen für die Schicht `presentation` bereitgestellt werden sollen. Eine Swing-Implementierung würde z.B. im Paket `mima.presentation.swing` zusammengefasst werden. Die GWT-basierte Implementierung der Schicht `presentation` wird im Abscnitt 7.4 beschrieben.

Die Programmiersprache Java besitzt für die im Systementwurf verwendeten UML-Konzepte, wie z.B. Klassen, Schnittstellen, Pakete und Operationen, entsprechende Sprachkonzepte. Insgesamt konnte der Systementwurf mit dem GWT sehr direkt implementiert werden, weil mit dem GWT in Java programmiert wird.

7.1.1 Exkurs: GWT-basierte GUI für eine existierende Java-basierte Domänenschicht

In dieser Arbeit entstand die Mind-Mapping-Anwendung „auf der grünen Wiese“, d.h die Domänen- und die Präsentationsschicht mussten von Grund auf neu entwickelt werden. In der Praxis ist aber folgendes Szenario denkbar. Eine Java-Swing-basierte Desktop-Anwendung soll durch eine GWT-basierte Web-Anwendung ersetzt werden. Die Anwendung wurde nach dem Separated-Presentation Muster entwickelt, so dass eine Präsentations- und

eine Domänenschicht existiert. Die Domänenschicht soll nun für die GWT-basierte Anwendung wiederverwendet werden. Soll die gesamte Anwendung, so wie in dieser Arbeit im Webbrowser laufen, darf die Domänenschicht keine Java-Sprachfeatures oder Java-Klassen verwenden, die von der emulierten JRE des GWTs nicht unterstützt werden. Beim Versuch die Domänenschicht wiederzuverwenden, sind die folgenden Fälle möglich:

- Die Domänenschicht ist GWT-kompatibel. In diesem Fall muss die Domänenschicht nur mit dem GWT-Compiler übersetzt werden.
- Die Domänenschicht ist nicht GWT-kompatibel, d.h. diese lässt sich nicht mit dem GWT-Compiler übersetzen. In diesem Fall muss die Domänenschicht entweder GWT-kompatibel umgeschrieben werden oder nicht im Webbrowser sondern auf dem Server laufen.

7.2 Schicht „domain“

Der Systementwurf für die Schicht `domain` konnte fast 1:1 implementiert werden. Die Schicht konnte durch ausschließliche Verwendung der GWT-Pakete `java.lang` und `java.util` implementiert werden. Da diese GWT-Pakete eine Emulation der Java-Pakete `java.lang` und `java.util` darstellen, kann die Schicht `domain` ohne Veränderungen auch in einer Java-Anwendung wiederverwendet werden.

Die Klasse `Frame` der Schicht `domain` ist eine Aufzählung. Die Objektmenge der Klasse `Frame` wird durch die vier Literale `CIRCLE`, `RECTANGLE`, `UNDERLINE` und `NONE` gebildet. In Java werden seit der Version 1.5 Aufzählungen durch den Typ `enum` direkt unterstützt. Vor Java 1.5 wurden Aufzählungen mittels des enum-Patterns [Schulz (2002)] realisiert. Da das GWT Java 1.5 noch nicht unterstützt, wird die Klasse `Frame` mit dem enum-Pattern realisiert.

7.3 Schicht „gui“

In Abschnitt 7.1 wurde bereits erwähnt, dass das GWT keine Implementierung für die Klasse `Canvas` enthält. In diesem Abschnitt werden verschiedene Ansätze zur Implementierung einer Klasse `Canvas` untersucht.

In Abschnitt 2.2.3 wurden verschiedene Vektorgrafiktechnologien, die von Webbrowsern unterstützt werden, beschrieben. Diese stellen die von `Canvas` zu implementierenden grafischen Primitive, wie z.B. das Zeichnen von Bézierkurven, zur Verfügung. In Abschnitt 3.4 wurde das „JavaScript Native Interface“ (JSNI) beschrieben, mit dem JavaScript-Code in eine

GWT-Anwendung integriert werden kann. Der generelle Ansatz der im Folgenden beschriebenen Ansätze zur Implementierung einer Klasse `Canvas` basiert auf der Verwendung einer Vektorgrafiktechnologie mittels JSNI.

In Abschnitt 2.1.2 wurde beschrieben, dass eine Ajax-Anwendung die Webbrowser Firefox und IE unterstützen muss. Eine Implementierung der Klasse `Canvas` auf Grundlage von VML ist daher nicht geeignet, weil VML nicht vom Webbrowser Firefox unterstützt wird.

SVG wird zwar vom Firefox, aber nicht vom IE nativ unterstützt. Für den IE müsste ein SVG-Plugin installiert werden. In Abschnitt 2.1.3 wurde beschrieben, dass eine Ajax-Anwendung nur auf nativen Browsertechnologien, d.h. nicht auf Plugins, basieren darf. SVG ist daher zur Implementierung einer Klasse `Canvas` auch nicht geeignet. Im Gegensatz zu VML gibt es für SVG aber schon eine `Canvas`-Implementierung. Robert Hanson stellt diese in der „GWT Widget Library“ zur Verfügung [Hanson (o.J.)]. Der Aufwand SVG mittels JSNI zu integrieren, würde daher entfallen.

Die JavaScript-Bibliothek jsGraphics von Walter Zorn unterstützt keine Bézierkurven. Die Bibliothek könnte sicherlich erweitert werden, doch müsste man dazu in JavaScript programmieren, was man jedoch durch die Benutzung des GWTs gerade vermeiden möchte.

Die Implementierung einer Klasse `Canvas` mittels des HTML-Elements Canvas scheint die vielversprechendste Lösung zu sein. Das HTML-Element Canvas wird vom Firefox nativ unterstützt. Der IE unterstützt das HTML-Element Canvas zwar nicht nativ, dieses kann aber durch die Verwendung von ExplorerCanvas emuliert werden. Das HTML-Element Canvas wurde bereits von Alexei Sokolov in das GWT integriert [Sokolov (o.J.)].

In dieser Arbeit wird aus den oben geschilderten Gründen die `Canvas`-Implementierung von Alexei Sokolov verwendet.

7.4 Schicht „presentation“

Der Systementwurf für die Schicht `presentation` konnte nahezu 1:1 implementiert werden. In Abbildung 7.2 sind die Pakete der Implementierung dargestellt. Die meisten Pakete sind aus dem Entwurfsmodell bekannt. Die Pakete `entrypoint` und `util` sind sehr GWT-spezifisch und waren daher im Entwurfsmodell nicht vorgesehen.

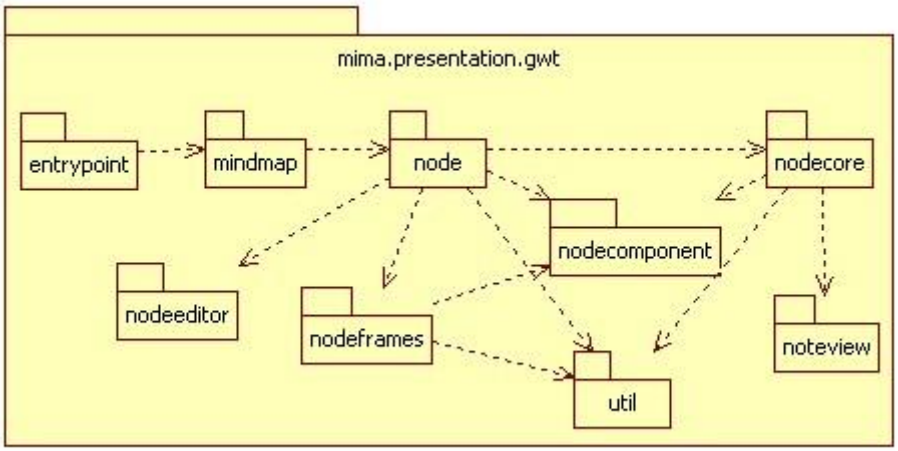

Abbildung 7.2: Implementierung der Schicht „presentation“

Das Paket `util` enthält die Klasse `WidgetUtilities`. `WidgetUtilities` definiert die statischen Methoden `getAbsoluteBottom(Widget widget)` und `getAbsoluteMiddle(Widget widget)`. Die Methoden berechnen den unteren Rand bzw. die vertikale Mitte von `widget`. Diese Methoden werden vom Paket `nodeframe`, `node` und `nodecore` benötigt. Die Methoden gehören eigentlich in die Klasse `com.google.gwt.user.client.ui.Widget` des GWTs, weil diese ausschließlich Methoden von `com.google.gwt.user.client.ui.Widget` für ihre Implementierung verwenden. Da es nicht zum guten Stil gehört, fremde Bibliotheksklassen zu ändern, wurden diese Methoden in der Klasse `WidgetUtilities` definiert.

Das Paket `entrypoint` definiert die Klasse `MiMa`, welche die Schnittstelle `com.google.gwt.core.client.EntryPoint` des GWTs implementiert. Diese Klasse bzw. genauer die zu implementierende Methode `onModuleLoad()` der Schnittstelle `com.google.gwt.core.client.EntryPoint` ist das Pendant zu der `main()`-Methode in klassischen Java-Anwendungen. Diese definiert den Einstiegspunkt des Programms.

7.5 MiMa

In diesem Abschnitt wird die Frage

> *Können mit dem Google Web Toolkit die Anforderungen an eine grafisch mittelmäßig anspruchsvolle 2D-Ajax-Anwendung umgesetzt werden?*

aus dem Einleitungskapitel beantwortet.

Die Anforderungen aus dem Kapitel 5 konnten mit dem GWT alle umgesetzt werden. In Abbildung 7.3 ist eine mit MiMa erstellte Mind-Map dargestellt. Zum Vergleich ist in Abbildung 7.4 eine mit FreeMind erstellte Mind-Map dargestellt. Ein Vergleich dieser beiden Mind-Maps zeigt, dass mit dem GWT grafisch mittelmäßig anspruchsvolle 2D-Anwendungen entwickelt werden können.

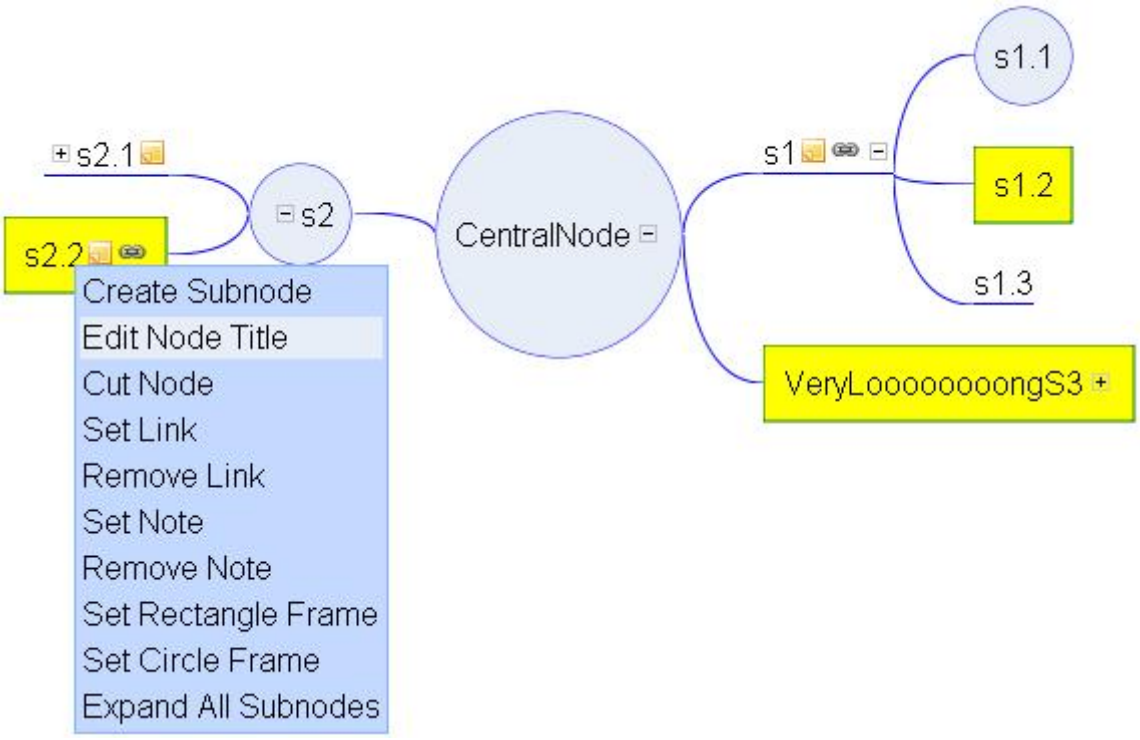

Abbildung 7.3: Benutzerschnittstelle von MiMa

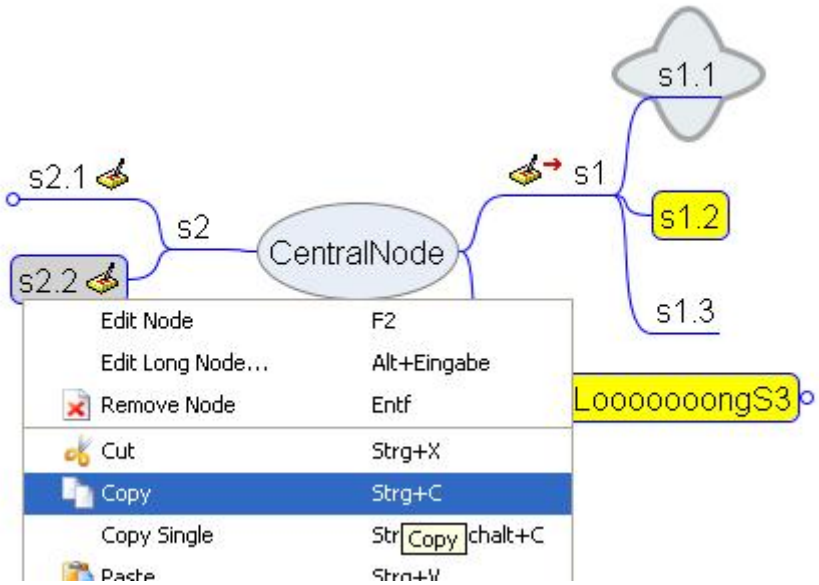

Abbildung 7.4: Benutzerschnittstelle von FreeMind

Obwohl alle Anforderungen umgesetzt werden konnten, kann die oben gestellte Frage nur mit Nein beantwortet werden, weil MiMa nur im Webbrowser Firefox richtig funktioniert. Im IE werden die Bézierkurven nicht dargestellt. Es scheint so, als ob ExplorerCanvas das HTML-Element Canvas im IE nicht zu 100% emulieren kann. Da MiMa nur im Firefox läuft, kann MiMa nicht als Ajax-Anwendung bezeichnet werden. Vektorgrafiken sind kein GWT-spezifisches Problem, sondern ein allgemeines Ajax-Problem. Es gibt heute noch keine Vektorgrafiktechnologie, die sowohl vom Firefox als auch vom IE nativ unterstützt wird.

8 Zusammenfassung

Im Rahmen dieser Arbeit wurde mit dem Google Web Toolkit (GWT) eine Mind-Mapping-Anwendung entwickelt. In Kapitel 5 wurden die Anforderungen an die Mind-Mapping-Anwendung definiert. Es wurden vor allem grafisch mittelmäßig anspruchsvolle Anforderungen, wie z.B. das Darstellen von Kreisen, Bézierkurven und Rechtecken auf der Benutzeroberfläche, definiert.

In Kapitel 6 wurde für die Anforderungen ein programmiersprachen-unabhängiger, objektorientierter Systementwurf nach State-of-the-Art-Entwurfsprinzipien, wie z.B. Modularisierung, Architekturmuster und Entwurfsmuster, erstellt. Es wurden die Architekturmuster Layers und Separated-Presentation verwendet. Als Entwurfsmuster wurde Beobachter zur Synchronisierung der Präsentations- und Domänenschicht verwendet. Des Weiteren wurden als Entwurfsmuster Dekorierer und Schablonenmethode verwendet.

In Kapitel 7 wurde die Implementierung des Systementwurfs mit dem GWT beschrieben und evaluiert. Der Systementwurf konnte fast 1:1 und sehr direkt implementiert werden. Die Fragestellung

> *Können mit dem Google Web Toolkit 2D-Ajax-Anwendungen nach den State-of-the-Art-Entwurfsprinzipien gebaut werden? (Entwicklersicht)*

dieser Arbeit, kann daher mit „Ja“ beantwortet werden.

Des Weiteren sollte die folgende Frage in dieser Arbeit beantwortet werden:

> *Können mit dem Google Web Toolkit die Anforderungen an eine grafisch mittelmäßig anspruchsvolle 2D-Ajax-Anwendung umgesetzt werden? (Benutzersicht)*

In Kapitel 2 wurde beschrieben, dass Ajax-Anwendungen nur als solche bezeichnet werden können, wenn diese sowohl im Webbrowser Firefox als auch im Internet Explorer funktionieren. Obwohl alle Anforderungen aus dem Kapitel 5 mit dem GWT umgesetzt werden konnten, kann die zweite Frage nur mit „Nein“ beantwortet werden, weil die entwickelte Mind-Mapping-Anwendung nur im Webbrowser Firefox richtig funktioniert. Zur Entwicklung grafisch mittelmäßig anspruchsvoller 2D-Ajax-Anwendungen werden Vektorgrafiktechnologien benötigt. Die Webbrowser Firefox und IE unterstützen heute noch keine gemeinsame Vektorgrafiktechnologie nativ. Vektorgrafiken sind somit kein GWT-spezifisches Problem, sondern ein allgemeines Ajax-Problem.

8.1 Ausblick

In dieser Arbeit wurden im Wesentlichen nur die Benutzerschnittstellenaspekte des GWTs untersucht. Weitere Aspekte, die noch evaluiert werden sollten, sind z.B. Sicherheit, Integrationsfähigkeit mit Legacy-Systemen und Persistenz.

Ob sich das Google Web Toolkit zur Entwicklung von Ajax-Anwendungen durchsetzen wird, ist nur schwer voraussehbar. Bereits heute gibt es vergleichbare Toolkits. Das Eclipse-Projekt Rich AJAX Platform (http://www.eclipse.org/proposals/rap/) und Echo2 (http://www.nextapp.com/platform/echo2/echo/) sind auch Ajax-Toolkits, in denen Ajax-Anwendungen in Java entwickelt werden. Eine vergleichende Evaluation dieser Toolkits wäre daher sinnvoll.

Ajax-Anwendungen werden heute in der Regel noch mit JavaScript-Frameworks wie Prototype, Script.aculo.us oder dem Dojo Toolkit entwickelt. Die Java-basierten Toolkits, wie das GWT, haben gegenüber diesen Frameworks den großen Vorteil, dass heute schon eine Vielzahl von Java-Werkzeugen zur Verfügung steht und, dass mit Java sehr strukturierte und wartbare Systeme gebaut werden können. Auch hier wäre es sinnvoll eine vergleichende Evaluation zwischen JavaScript-basierten und Java-basierten Ajax-Toolkits durchzuführen, um die Vor- und Nachteile der jeweiligen Ansätze herauszuarbeiten.

Literaturverzeichnis

[Buschmann u. a. 2000] BUSCHMANN, Frank ; MEUNIER, Regine ; ROHNERT, Hans ; SOMMERLAD, Peter ; STAL, Michael: *Pattern-orientierte Softwarearchitektur*. München : Addison-Wesley, 2000

[Buzan und Buzan 2006] BUZAN, Tony ; BUZAN, Barry: *The Mind Map Book*. BBC Active, 2006

[Cockburn 2001] COCKBURN, Alistair: *Writing Effective Use Cases*. Addison-Wesley, 2001

[Crane u. a. 2006] CRANE, Dave ; PASCARELLO, Eric ; JAMES, Darren: *AJAX in action*. Dt. Übers. Addison-Wesley, 2006

[Flanagan 2002] FLANAGAN, David: *JavaScript*. 2. Auflage. 2002

[Fowler 2003] FOWLER, Martin: *Patterns of enterprise application architecture*. Boston Mass. : Addison-Wesley, 2003 (The Addison-Wesley signature series)

[Fowler 2005] FOWLER, Martin: *Refactoring*. Dt. Übers. Addison-Wesley, 2005

[Fowler o.J.] FOWLER, Martin: *Development of Further Patterns of Enterprise Application Architecture*. o.J.. – URL http://www.martinfowler.com/eaaDev/

[Gamma u. a. 1996] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Entwurfsmuster*. 5. korrigierter Nachdruck. Addison-Wesley, 1996

[Garrett 2005] GARRETT, Jesse J. ; PATH, Adaptive (Hrsg.): *AJAX: A New Approach to Web Applications*. 2005. – URL http://adaptivepath.com/publications/essays/archives/000385.php

[Google 2006] GOOGLE ; GOOGLE (Hrsg.): *Google Web Toolkit*. 2006. – URL http://code.google.com/webtoolkit/

[Hanson o.J.] HANSON, Robert: *GWT Widget Library*. o.J.. – URL http://gwt-widget.sourceforge.net/

[Larman 2005] LARMAN, Craig: *Applying UML and patterns*. 3. Auflage. Upper Saddle River, NJ : Prentice Hall [u.a.], 2005

[Müller u. a. o.J.] MÜLLER, Jörg ; POLANSKY, Daniel ; NOVAK, Petr ; FOLTIN, Christian ; POLIVAEV, Dimitri: *FreeMind*. o.J.. – URL http://freemind.sourceforge.net/wiki/index.php/Main_Page

[OpenAjaxAlliance 2006] OPENAJAXALLIANCE: *OpenAjax Alliance White Paper*. 2006. – URL http://www.openajax.org/OpenAjax%20Alliance%20White%20Paper.pdf

[O'Reilly 2005] O'REILLY, Tim ; O'REILLY (Hrsg.): *What Is Web 2.0*. 2005. – URL http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[Schulz 2002] SCHULZ, Holger: Das enum-Pattern. In: *JavaSPEKTRUM* (2002), Nr. 11. – URL http://www.sigs.de/publications/js/2002/11/schulz_JS_11_02.pdf

[Sokolov o.J.] SOKOLOV, Alexei: *Canvas Widget*. o.J.. – URL http://gwt.components.googlepages.com/canvas

[W3C 1998] W3C: *Vector Markup Language (VML)*. 1998. – URL http://www.w3.org/TR/NOTE-VML.html

[W3C 2002] W3C ; W3C (Hrsg.): *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*. 2002. – URL http://www.w3.org/TR/xhtml1/

[W3C 2003] W3C: *Scalable Vector Graphics (SVG)*. 2003. – URL http://www.w3.org/Graphics/SVG/

[W3C 2004] W3C ; W3C (Hrsg.): *Document Object Model (DOM) Level 3 Core Specification*. 2004. – URL http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/

[W3C 2006] W3C ; W3C (Hrsg.): *Cascading Style Sheets, level 2 revision 1*. 2006. – URL http://www.w3.org/TR/CSS21/

[W3Schools o.J.] W3SCHOOLS ; SCHOOLS, W3 (Hrsg.): *Browser Statistics*. o.J.. – URL http://www.w3schools.com/browsers/browsers_stats.asp

[WHATWG 2007] WHATWG: *Web Applications 1.0*. 2007. – URL http://www.whatwg.org/specs/web-apps/current-work/

[Zorn o.J.] ZORN, Walter: *http://www.walterzorn.de*. o.J.. – URL http://www.walterzorn.de/jsgraphics/jsgraphics.htm

A CD

Auf der CD befindet sich

- das Eclipse-Projekt. Das Projekt enthält den Quellcode, die Unit-Tests, die Javadoc-Dokumentation, die verwendeten Bibliotheken und die UML-Diagramme dieser Arbeit. Zum Öffnen der UML-Diagramme wird die Software StarUML (http://staruml.sourceforge.net/en/) benötigt.
- diese Arbeit als PDF-Dokument

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 29. März 2007	
Ort, Datum	Unterschrift