Universität Hamburg

Fakultät für Mathematik, Informatik und Naturwissenschaften

Department Informatik

Arbeitsbereich Softwaretechnik

Bakkalaureatsarbeit

Migration einer Eclipse Rich Client Anwendung in eine webbasierte Ajax Anwendung am Beispiel des Lernassistenten LAssi

Betreuer: Dr. Guido Gryczan

Marc Päpper

marc.paepper@informatik.uni-hamburg.de

Studiengang Dipl. Informatik, Nebenfach: Psychologie

Matr.-Nr. 5715362

Inhaltsverzeichnis

Abbildungsverzeichnis

Quelltextverzeichnis

1) Einleitung

Diese Bakkalaureatsarbeit ist im Rahmen des Projektes „Objektorientierte Softwareentwicklung“ im Hauptstudium, welches im Wintersemester 2006/2007 sowie im Sommersemester 2007 stattfand und von Guido Gryczan und Heinz Züllighoven geleitet wurde, entstanden.
Das Projekt beschäftigt sich mit der Weiterentwicklung der Lernsoftware LAssi[1] (Lern-Assistent), die bereits seit 1998 aus Modellversuchen der Hamburger Schulbehörde zum Lernen mit Notebooks in Schulen entwickelt wurde.

Im ersten Semester des Projektes ging es vor allem darum, die vorhandene Funktionalität von LAssi zu erweitern, um den Schülern, die das Produkt verwenden, einen Mehrwert an Möglichkeiten zu bieten. Zu diesem Zweck haben sich die Studenten - zu denen auch der Autor dieser Arbeit gehört - in das LAssi Projekt eingearbeitet und haben dann im Plenum Möglichkeiten diskutiert, um LAssi zu erweitern. Es haben sich im Laufe der Diskussion fünf unterschiedliche Erweiterungen herauskristallisiert, so dass sich Gruppen zu je sechs bis neun Studierenden fortan um folgende Features gekümmert haben:

a) Ein Kalenderwerkzeug, um den Schülern die Möglichkeit zu bieten, ihre Termine und insbesondere ihren Stundenplan in LAssi zu verwalten
b) Eine Mal- und Zeichenunterstützung mit der es möglich ist, kleine Skizzen in LAssi anzufertigen
c) Einen Vokabeltrainer mit dem die Schüler ihre Wortschätze verwalten können und der sie in einer spielerischen Art und Weise beim Erlernen der Vokabeln unterstützt.
d) Eine Audio- und Videounterstützung, die es ermöglicht Sprachdateien und Filmsequenzen in Karteikarten einzubinden und die eine Playlist Verwaltung bietet
e) Einen Formeleditor, der es den Schülern gestattet, naturwissenschaftliche Formeln ansprechend grafisch darzustellen und in bestehende Karteikarten einzubinden

1 Für eine genauere Beschreibung von LAssi siehe Kapitel 2.1

Ich habe dabei in der Gruppe des Zeichenwerkzeugs mitgearbeitet, welches im Laufe des Wintersemesters zu einem großen Teil implementiert wurde.
Im zweiten Semester stand dann im Vordergrund, die bisher entwickelten Erweiterungen so zu modifizieren, dass man diese problemlos in die neue LAssi Version 2.1 einbinden kann. Einige der Studierenden haben nur im Wintersemester an dem Projekt teilgenommen, so dass auf Grund der personellen Schwächung im Projekt diskutiert wurde, sich auf eine bestimmte Erweiterung zu konzentrieren, oder alle entwickelten Erweiterungen weiterzuführen. Es stellte sich schließlich heraus, dass die Mehrheit für die Weiterentwicklung aller Tools war, so dass am Ende alle Erweiterungen mit Ausnahme des Formeleditors weiterentwickelt wurden.

Im Laufe dieser Bakkalaureatsarbeit möchte ich mich nun damit beschäftigen, inwieweit es möglich ist, eine Eclipse Rich Client Anwendung wie LAssi in eine webbasierte Ajax Anwendung zu migrieren. Dazu werde ich auf die beteiligten Technologien und den betrachteten Kontext eingehen und im Laufe der Arbeit untersuchen, welche Werkzeuge vorhanden sind, um diese Aufgabe zu bewerkstelligen und welche möglichen Probleme es dabei gibt.
Ich werde mich dabei auf den Ausschnitt des LAssi Zeichenwerkzeugs beschränken und meine Vorgehensweise und Erfahrungen schildern, dieses als Webanwendung mit Ajax zu realisieren.

An dieser Stelle erläutere ich kurz den weiteren Verlauf meiner Bakkalaureatsarbeit. Im zweiten Kapitel meiner Arbeit möchte ich dem Leser die verwendeten Begriffe und Technologien, die für das weitere Vorgehen relevant sind, näher vorstellen, um die Art und den Umfang des Migrationsprozesses zu verdeutlichen. Dazu werde ich zunächst das LAssi Projekt genauer präsentieren, um klarzumachen, in welchem Umfeld das von meiner Gruppe entwickelte Zeichenwerkzeug operiert, das dann im vierten Kapitel näher erläutert wird. Um auch den technischen Kontext zu verdeutlichen, gehe ich dabei kurz auf die Eclipse Rich Client Plattform ein, um anschließend klar zumachen, worin die Herausforderung bei einer Migration liegt. Ich stelle dann den Zielkontext im Sinne von Ajax und des Web 2.0 vor und zeige Unterschiede und Gemeinsamkeiten zum Ursprungskontext auf.
Im dritten Kapitel spanne ich dann den Bogen zwischen dem Ursprungskontext und dem Zielkontext, indem ich das Google Web Toolkit (GWT) präsentiere, welches ich zum Zwecke

der Migration benutzen werde. Mit Hilfe dieses Tools ist es möglich, seine Anwendung in einer Hochsprache (Java) in der IDE[2] seiner Wahl zu entwickeln, um sich vom GWT dann den zum Programm korrespondierenden JavaScript Code generieren zu lassen, der in allen gängigen Browsern funktioniert.

Das Ziel des vierten Kapitels ist es, dem Leser, der an dieser Stelle das Ziel der Arbeit vor Augen haben sollte, das Konzept und die Struktur des programmierten Zeichenwerkzeugs näher zu bringen, um ein genaueres Gefühl des Werkzeugs zu erlangen und sich den erforderlichen Migrationsaufwand besser vorstellen zu können.

Nachfolgend werde ich im fünften Kapitel das von mir gewählte Vorgehen bei der Migration unter Verwendung des Google Web Toolkits vorstellen und in mehreren Schritten präsentieren, wie ich vorgegangen bin und welche Probleme und deren Lösungen dabei auftraten.

Zum Ende meiner Arbeit reflektiere ich meine im Laufe des Prozesses gemachten Erfahrungen und weise auf Probleme, Chancen und mögliche Verbesserungen hin.

2 Integrated Development Environment – Entwicklungsumgebung wie z.B. Eclipse (siehe auch Kapitel 2.2)

2) Begriffsabgrenzung

Dieses Kapitel dient dazu, dem Leser mit den in der Arbeit verwendeten Begriffen und Technologien vertraut zu machen, um damit alle Voraussetzungen zu schaffen, den Kern der Arbeit verstehen zu können.
Es wird dazu zunächst das Programm LAssi beschrieben, um den Kontext des Zeichenwerkzeugs zu präsentieren (auf das Zeichenwerkzeug selber wird im vierten Kapitel näher eingegangen). Anschließend wird die Eclipse Rich Client Plattform beschrieben, die den Ausgangspunkt der Migration definiert, da LAssi und folglich auch das Zeichenwerkzeug mit Hilfe der Eclipse Rich Client Plattform entwickelt wurde. Gegen Ende des Kapitels wird abschließend der Zielkontext Ajax als Technologie moderner Webanwendungen beschrieben, um klarzumachen wohin die Reise gehen soll. Durch die detaillierte Vorstellung von Ajax soll dem Leser bewusst werden, dass es anders als in Eclipse wesentlich schwieriger ist, eine Ajax Anwendung strukturiert aufzubauen, da es viele Unterschiede in den verschiedenen Browsern gibt. Um dieses Hindernis zu umgehen wird dann im dritten Kapitel dieser Arbeit das Google Web Toolkit als ein Werkzeug vorgestellt, das das Programmieren auf dieser unbequemeren Ebene umgehen soll.

2.1) LAssi

2.1.1) Grundlegende Konzeption

LAssi ist die Abkürzung für Lern-Assistent bzw. The Learner's Assistant und wird im Rahmen eines Public-Private-Partnership-Projektes zwischen IBM und der Hamburger Schulbehörde mit objektorientierten Techniken entwickelt.

Die Idee von LAssi ist es, Anwendern eine digitale Lernumgebung zu bieten, so dass sie ihr erworbenes Wissen selber verwalten, ordnen und strukturieren können. So wurde dieser Lernassistent gemeinsam von Pädagogen und Informatikern entwickelt und dabei so konzipiert, dass er auf einen USB-Stick passt und von diesem Stick direkt ausgeführt werden kann, ohne vorher etwas zu installieren. Das ermöglicht dem Lernenden, all seine wichtigen Lerndateien immer dabei zu haben und an jedem beliebigen Arbeitsplatz ohne Einschränkung

und ohne Zeitverlust sofort einsetzen zu können.
Dabei soll LAssi die Individualität des Lernenden unterstützen und zahlreiche Lernwerkzeuge auf einer Plattform integrieren von denen der Lernende sich die für ihn am besten geeigneten auswählen kann.
Damit folgt LAssi modernen Ergebnissen der Lernpsychologie, indem es nicht die Wissensvermittlung und die Belehrung in den Vordergrund stellt, sondern die aktive Wissensaneignung durch die systematische Organisation eigenen Wissens (Vgl. [LAP06]).

2.1.2) Beschreibung des Kernprogramms

Im folgenden beziehe ich mich auf die LAssi Version 2.1, die im Projekt im Sommersemester 2007 verwendet wurde und für die im Laufe des Projektes Erweiterungen entwickelt wurden.

Das wohl wichtigste Konzept an LAssi ist der Desktop und die Karteikarten. Der Desktop stellt eine Arbeitsfläche dar, wie man sie auch vom „Windows Desktop“ her kennt und ist somit eine Metapher für den Schreibtisch der Nutzer. Die Schüler haben die Möglichkeit, unterschiedliche Dinge auf einem Desktop anzulegen, die dann normalerweise in Form einer Karteikarte repräsentiert werden. Einer Karte, die auf dem Desktop liegt, können unterschiedliche Attribute – wie etwa ihre Farbe – zugewiesen werden und man kann die Karte frei auf dem Desktop hin- und herschieben. Darüber hinaus bietet der Desktop zahlreiche Möglichkeiten, um seine Karteikarten zu organisieren. Die vielleicht intuitivste dabei ist, mehrere Karten zu einem Stapel zu gruppieren, so dass man den gesamten Stapel als ein Objekt verschieben kann. Es ist dabei grafisch zu erkennen, wie viele Karten sich in dem Stapel befinden, die oberste Karte wird angezeigt und man kann mit Hilfe von Pfeilsymbolen durch den Stapel blättern.
Etwas radikaler als das Stapeln von Karten ist das Zusammenfassen von Karten, denn hierbei wird aus mehreren Karte eine einzelne Karte, die die Inhalte der zusammengefassten Karten in sich vereinigt. Man kann aber eine Karte durch die „Aufteilen“ Funktion auch wieder in verschiedene Karten feinerer Granularität aufteilen.
Ein weiteres Werkzeug, um Karten in LAssi zu ordnen, ist der so genannte Sortierkasten. Dieser wird auch als eigene Karteikarte dargestellt und bietet eine Art Tabellenansicht, so dass man verschiedene Karten in ein Zeilen- und Spaltenmuster einordnen kann. Die Zeilen und

Spalten kann man dabei benennen, so dass sich ein Sortierkasten zum Beispiel sehr gut eignet, um sich ein Thema für ein Referat zu strukturieren. Als Vereinfachung des Sortierkastens kann man den Pro-/Kontrakasten nennen, denn dieser besteht nur aus einer Zeile und zwei Spalten, so dass man die Argumente in Form von Pro-Karteikarten in die eine Box und die Argumente in Form von Kontra-Karteikarten in die andere Box legen kann.
Die Pinnwand ermöglicht es dem Lerner, verschiedene Karten in Bezug zueinander zu setzen, indem er sie mit Pfeilen verbindet und auf einer Fläche zueinander anordnet und die Kompetenzamöbe sorgt dafür, dass man einen Überblick über seine Leistungen grafisch darstellen kann, um zu beobachten wie man sich in gewissen Dingen entwickelt hat. Die Kompetenzamöbe kann also beispielsweise dazu genutzt werden, seinen Notenverlauf in den unterschiedlichen Fächern einzugeben und so auf einen Blick erkennen zu können, in welchen Bereichen man sich verbessert hat und in welchen Bereichen man eventuell etwas mehr Einsatz nötig hätte.
Um die einzelnen Karteikarten zu modifizieren und mit Inhalten zu füllen, steht ein eigener Editor zur Verfügung, der sich öffnet, wenn man doppelt auf eine Karte klickt. Der Bildschirm wird dann geteilt, so dass man in der einen Hälfte weiterhin einen Ausschnitt des aktiven Desktops sieht und in der anderen Hälfte die Bearbeitungsansicht der entsprechenden Karte. In dem Editorfenster der Karteikarten gibt es einen Abschnitt für die Überschrift der Karte, einen Bereich zum Anlegen von Text- sowie Bildeinträgen und am Ende die Möglichkeit, Dateianhänge hinzuzufügen. Bei LAssi wird stets darauf geachtet, eine möglichst einfache und intuitive Bedienung zu ermöglichen, so dass man Informationen nicht nur selber in der Karte eintippen kann, sondern auch per Drag & Drop aus geöffneten Dokumenten - wie etwa einer Homepage - direkt in die Karte ziehen kann.

2.2) Eclipse Rich Client Plattform

In diesem Abschnitt wird die Eclipse Rich Client Plattform (Eclipse RCP) als Ursprungskontext beschrieben, in der das zu Grunde liegende Programm entwickelt wurde. Die Intention ist dabei nicht, die gesamten Konzepte der Eclipse RCP vorzustellen, da dieses bereits Gegenstand vieler Bücher ist, sondern dem Leser, der die Plattform möglicherweise noch nicht kennt, einen Einblick in die Möglichkeiten dieser Plattform zu geben.

2.2.1) Entstehung der Eclipse Rich Client Plattform

Die Eclipse RCP ist eine Weiterentwicklung von Eclipse. Eclipse ist eine so genannte IDE (Integrated Development Environment), die eine Umgebung für die produktive Entwicklung von Software durch Unterstützung mit zahlreichen Hilfsmitteln und Werkzeugen darstellt. In dieser Hinsicht wurde Eclipse von IBM entwickelt, um die damalige IDE Visual Age von IBM zu ersetzen. Im November 2001 wurde aus der Eclipse Plattform dann ein Open Source Projekt, so dass in der folgenden Zeit von vielen Seiten Erweiterungen in die Plattform eingeflossen sind (Vgl. [ECL07] sowie [DAU07] Kapitel 1.1). Bis zur Version 2.1 bestand Eclipse in dieser Form als IDE für unterschiedliche Programmiersprachen – besonders jedoch für die Sprache Java. Eine Besonderheit bei Eclipse ist dabei, dass die unterschiedlichen Funktionalitäten als Plugins entwickelt werden. Dadurch besteht die Möglichkeit, die Eclipse IDE jederzeit durch weitere Plugins zu erweitern, um so zusätzliche Werkzeuge zu integrieren. Die Flexibilität dieser Plugin Architektur ermöglichte mit der Version 3.0 die Umstrukturierung der Eclipse Plattform von einer IDE hin zu einer allgemeineren Plattform für die Entwicklung von Rich Client Anwendungen[3]. Damit wurde die ursprüngliche Eclipse IDE selbst zu einer spezifischen Rich-Client-Anwendung. Zusätzlich wurde mit der Version 3.0 eine Umstellung der vormals proprietären Ablaufumgebung auf die offene OSGi[4]-Plattform vollzogen. OSGi beinhaltet eine standardisierte Ablaufumgebung für Java-Module (Bundles genannt), deren besonderer Vorteil das so genannte Hot Plugging ist. Hot Plugging bezeichnet den Austausch von Java-Modulen im laufenden Betrieb ohne den Server, auf dem die Plattform läuft, neu starten zu müssen (Vgl. [DAU07] Kapitel 1.1 sowie 3.2).

3 Was genau eine Rich Client Anwendung ist, wird im nächsten Unterpunkt erläutert.

4 OSGi ist die Abkürzung für Open Service Gateway Initiative

2.2.2) Merkmale einer Rich Client Anwendung

Der Begriff Rich Client Anwendung ist nun bereits mehrmals gefallen und in diesem Abschnitt werde ich auf die wesentlichen Merkmale einer solchen Anwendung eingehen. Diese sind State-of-the-Art-Benutzungsoberfläche[5], Notlaufeigenschaften und Server-managed clients (nach [DAU07] S.3-5).

Unter der Eigenschaft **State-of-the-Art-Benutzungsoberfläche** versteht man dabei, dass die Anwendungsoberfläche der Betriebssystemoberfläche in nichts nachsteht, sondern sich in angemessener Weise eingliedert, so dass die Bedienung für den Benutzer intuitiv ist. In der Eclipse Rich Client Plattform wird das durch die Verwendung des Standard Widget Toolkits (SWT) ermöglicht, das jeweils die GUI-Elemente des jeweiligen Betriebssystems, unter dem die Anwendung läuft, verwendet. So hat die Anwendung, wenn sie auf dem Betriebssystem Microsoft Windows ausgeführt wird, das Look and Feel einer Windows Anwendung und wenn sie auf dem Apple Betriebssystem Mac OS X ausgeführt wird, das bekannte Apple Look and Feel.

Mit dem Begriff **Notlaufeigenschaften** ist gemeint, dass die Anwendung auch dann funktioniert, wenn sie keine Verbindung zum Server hat und damit lokal ausgeführt wird. Folglich müssen sowohl die Daten als auch die Anwendungslogik in einer Rich Client Anwendung lokal gehalten werden können. Bei einer Verbindung zum Server muss dann ein Synchronisationsmechanismus dafür sorgen, dass die lokalen Daten und die Daten auf dem Server miteinander abgeglichen werden. Als gutes Beispiel dient hier ein Terminmanagementsystem, welches auf einem mobilen PDA ausgeführt wird. Der Benutzer möchte auch Termine eintragen können, wenn er gerade keine Internetverbindung besitzt, aber er möchte die Termine später auch nicht manuell erneut eintragen müssen.

Der dritte Punkt der **Server-managed Clients** betrifft die Abschottung des Nutzers von der Installation und der Konfiguration der Software. Die Software sollte vielmehr automatisch installiert, aktualisiert und konfiguriert werden. Zu diesem Punkt halte ich das LAssi Konzept des USB Sticks für ein gutes Beispiel, da der Nutzer einen USB Stick erhält, welchen er auf einem beliebigen System sofort und ohne Installation ausführen kann.

5 Im [DAU07] wird der Begriff State-of-the-Art-Benutzeroberfläche verwendet. Da Benutzeroberfläche keine gelungene Übersetzung für GUI (graphical user interface) darstellt, verwende ich im folgenden den Begriff Benutzungsoberfläche.

2.2.3) Merkmale der Eclipse Rich Client Plattform

Wie bereits eingangs erwähnt, baut die Eclipse RCP auf einer Plugin Architektur auf. Der eigentliche Kern von Eclipse ist dabei relativ klein und nur dafür zuständig, Plugins zum Laufen zu bringen. Die ursprüngliche Eclipse IDE ist selber eine durch Plugins erstellte Rich Client Anwendung, die durch den Eclipse Kern zum Laufen gebracht wird. Ein Plugin stellt eine klar abgegrenzte Funktionalität einer Anwendung dar, so dass eine Anwendung aus mehreren unterschiedlichen Plugins bestehen kann. Um die unterschiedlichen Plugins zu verknüpfen und in einer Anwendung zu integrieren, werden Erweiterungspunkte (extension points) definiert, die spezielle Punkte darstellen, an denen ein Plugin erweitert werden kann.

Jede Anwendung hat ein spezielles Plugin, das die Anwendung konfiguriert und als Einstiegspunkt dient. Dieses spezielle Plugin wird Anwendungs-Plugin genannt. Wenn man ein Plugin etwas genauer betrachtet, so besteht es in der Regel aus in Paketen (Packages) organisierten Java Klassen, die die Funktionalität beschreiben, einer OSGi-Manifest Datei und einem Plugin-Manifest. Das OSGi-Manifest legt fest, welche anderen Plugins von diesem Plugin benötigt werden und welche Pakete mit Java Klassen des Plugins für die anderen Plugins sichtbar sein sollen. Das Plugin-Manifest beschreibt die schon erwähnten Erweiterungspunkte, indem es angibt, an welchen Erweiterungspunkten das Plugin andere Plugins erweitert und an welchen Stellen wiederum es selber erweitert werden kann (Vgl. [DAU07] Kapitel 3 und 4).

Abbildung 1 zeigt die Trennung der Eclipse IDE von der Eclipse RCP. Der dunkelbraun markierte Bereich stellt die Techniken zusammen, die die Grundlage jeder Eclipse Rich Client Anwendung sind. Die Hilfe, Update und Text Funktionalitäten können optional zusätzlich genutzt werden. Weiß markiert ist die als eigenständige Rich Client Anwendung entwickelte IDE, die aus unterschiedlichen Plugins besteht.

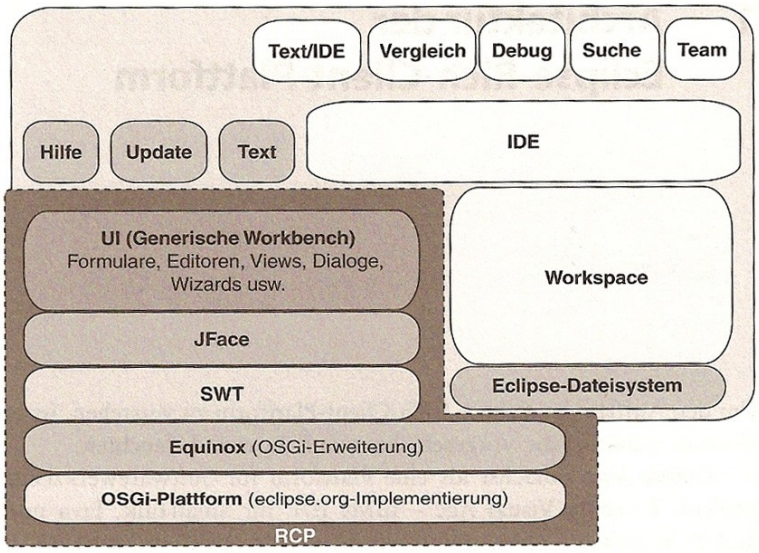


Abbildung 1: Die Eclipse Plattform. Aus: [DAU07] S.12

Das interessante an der Eclipse RCP ist also, dass die Eclipse IDE selber als eigenständige Rich Client Anwendung läuft und diese Anwendung dem Entwickler dazu dient, eigene Rich Client Anwendungen zu programmieren und dabei auf viele bereits bestehende Lösungen bauen zu können (Formulare, Editoren, Views, Dialoge, Wizards usw.). Weiter können die Anwendungen innerhalb der Eclipse IDE als eigene Anwendung gestartet und getestet werden.

Durch dieses Konzept ist auch die Architektur von LAssi sehr flexibel gehalten, da Werkzeuge in LAssi durch kompatible Plugins einfach dem Gesamtsystem hinzugefügt werden können. So konnten unterschiedliche Gruppen parallel an verschiedenen Werkzeugen arbeiten, die dann dem Gesamtsystem hinzugefügt werden können. Wie dem Leser unter Umständen aufgefallen ist, entspricht LAssi derzeit noch nicht vollständig dem Punkt der Notlaufeigenschaften, da dieser voraussetzt, dass eine Rich Client Anwendung auch mit einem Server kommuniziert. Wenn man sich jedoch die Road Map für die geplante LAssi Entwicklung anschaut, so erkennt man, dass sich das Projekt in diese Richtung weiterentwickelt, indem es beispielsweise das Wissensnetz von Schulen in LAssi einbeziehen möchte (Vgl. [LAS06]).

2.3) Ajax

2.3.1) Ajax im Kontext des Web 2.0

Der Name Ajax im Sinne dieser Arbeit kommt nicht wie ein interessierter Leser denken mag von einem Asteroiden, einer CIA-Operation oder eines Herrschers aus dem antiken Griechenland auch wenn es für diese Begriffe ebenfalls den Ausdruck Ajax gibt, sondern ist vielmehr die Abkürzung der Kombination verschiedener Technologien. So ist Ajax „ein Apronym[6] für die Wortfolge **A**synchronous **J**avaScript **a**nd **X**ML“ [WIK07] und setzt sich folglich maßgeblich aus zwei schon vorher bestehenden Technologien nämlich JavaScript und XML zusammen ohne selber eine neue Technologie oder gar eine Programmiersprache darzustellen. Trotzdem kann man sagen, dass sich durch das im Zuge des Web 2.0 in Mode gekommenen Wortes Ajax die Weblandschaft gehörig verändert hat. Zu den Schlüsseltechnologien des Web 2.0 zählen dabei neben Ajax noch folgende drei weitere wichtige Dinge: Mashup (Integration unterschiedlicher Anwendungen unter einer einheitlichen Oberfläche), Social Network Analysis (Analyse von Information und Wissen des sozialen Netzwerks vieler Personen) und die Kollektive Intelligenz (Generierung von Inhalten durch das Zusammenspiel vieler Personen) [Vgl. STE07 S.11-12]. Die Aufmerksamkeit für Ajax lieferte vor allem ein Aufsatz des Amerikaners Jesse James Garret mit dem Titel: „Ajax: A New Approach to Web Applications“ Anfang des Jahres 2005 [GAR05],

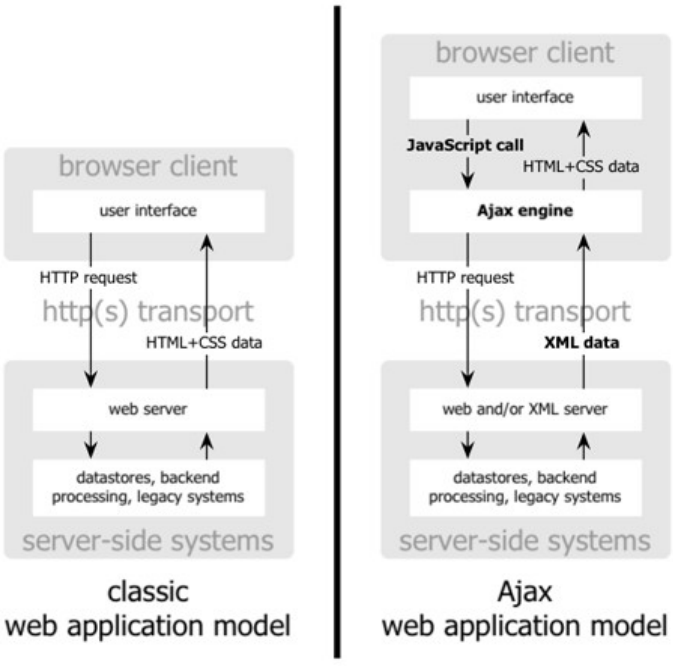


Abbildung 2: Vergleich des herkömmlichen Modells einer Webanwendung mit dem Modell einer Ajax Webanwendung. Aus [GAR05]

6 Ein Kunstwort, das aus den Anfangsbuchstaben mehrerer Wörter zusammengesetzt wird, aber auch selber als eigenständiger Begriff verwendet wird.

da dieser Aufsatz zum ersten Mal die unterschiedlichen Technologien unter dem Namen Ajax vereint hat.

Warum hat Ajax nun dazu beigetragen, dass sich ganz neue Möglichkeiten für Webanwendungen ergeben haben? Wie im linken Teil der Abbildung 2 zu erkennen ist, wird im herkömmlichen Modell einer Webanwendung bei jeder lokalen Anfrage (engl. „Request“) eines Nutzers eine Anfrage an den Webserver weitergereicht, der die Anfrage verarbeitet und schließlich als Antwort eine HTML[7]-Datei zurücksendet. Der Nutzer muss bei jeder Anfrage also warten, bis der Webserver mit seinen Berechnungen fertig ist und die Antwort liefert. Dabei wird bei jeder Anfrage die komplette Seite neu geladen, so dass es für den Nutzer offensichtliche Unterbrechungen gibt, in denen er eine leere Seite vorfindet.
Im Gegensatz dazu gibt es im Ajax Modell einer Webanwendung eine weitere Instanz: Die Ajax Engine. Die Ajax Engine liegt auf der Client Seite und operiert zwischen dem User Interface auf der einen und dem Webserver auf der anderen Seite. Der Hauptvorteil bei dieser Variante ist, dass die Webseite nicht immer komplett neu geladen wird. Vielmehr stellt die Ajax Engine wenn sie eine Anfrage des Nutzers erhält eine Anfrage an den Webserver und erhält die Antwort von ihm zum Beispiel[8] im XML Format (Vgl. Abbildung 2 rechte Seite) . Das DOM[9] Konzept ermöglicht es dann die so gewonnene Information an beliebiger Stelle der Seite einzufügen und somit sogar die Struktur der Seite dynamisch zu verändern.

Durch die Kombination von asynchronem JavaScript und XML entstanden in der Folgezeit Webanwendungen, wie sie vorher nicht möglich und vielleicht sogar kaum vorstellbar waren, da der Benutzer nun häufig den Eindruck hat, dass die Seiten wesentlich flexibler, direkter und dynamischer wirken. Ein klassisches und sehr bekanntes Beispiel für eine erste große Ajax-Anwendung ist Google Maps [GOO07], welches dem Benutzer Karten der ganzen Welt liefert und es ihm ermöglicht in den Karten schnell und ohne lange Ladezeiten zu navigieren. Auch wenn der Ajax Trend sicherlich noch nicht vorbei ist, kann man in einem kleinen Rückblick vielleicht schon sagen, dass das Aufkommen des Begriffes eine günstige Zeit erwischt hat, da relativ zeitgleich auch userzentrierte Anwendungen, die den Benutzer in den

7 Die HyperText Markup Language ist eine Auszeichnungssprache, um Texte, Bilder und vieles mehr darzustellen

8 Weitere Möglichkeiten sind reiner Text und das JSON Format (**J**ava**S**cript **O**bject **N**otation)

9 Zu DOM siehe Kapitel 2.3.2) Technischer Hintergrund von Ajax

Mittelpunkt stellen und mehr und mehr von vom Benutzer generierten Inhalten leben (user generated content) aufkamen, so dass sich der Nutzer durch die direktere Art von Ajax zusätzlich einbezogener auf der Homepage fühlt.

Interessant für meinen Migrationsansatz ist in diesem Zusammenhang, dass sich die Webanwendungen zunehmend an gewöhnliche Desktoplösungen anpassen, denn mehr und mehr entstehen solche Anwendungen, die auf den ersten Blick kaum mehr etwas mit einer Webanwendung zu tun haben – abgesehen davon, dass sie im Browser[10] laufen. Ein besonders gutes Beispiel dafür ist die Seite Ajax Launch [AJA07], die ein komplett online verfügbares, mit Ajax Technologie arbeitendes Office Paket zur Verfügung stellt. So kann man etwa mit ajaxWrite Textdokumente erstellen und mit ajaxXLS Tabellenkalkulationen durchführen, um nur zwei Anwendungen von Ajax Launch zu erwähnen. Warum halte ich das für interessant? Mit zunehmender Abnahme des Unterschiedes zwischen Desktopanwendungen und Webanwendungen gewinnt der Migrationsansatz zunehmend an Bedeutung. Denn es ist nicht schwer einzusehen, dass es für Firmen interessant ist, eine Desktopanwendung zu entwickeln und diese mit möglichst wenig Aufwand in eine webbasierte Lösung portieren zu können, die eine möglichst identische Funktionalität bietet. Das ist natürlich nur möglich, wenn die webbasierte Lösung die Funktionalität der Desktoplösung abbilden kann, so dass die abnehmende Distanz beider Lösungen erst eine tatsächliche Migration erlaubt.

2.3.2) Technischer Hintergrund von Ajax

Wie bereits erwähnt, ist Ajax keine neue Technologie, sondern vielmehr ein Begriff, der verschiedene Technologien und ihre gemeinsame Wirkung zusammenfasst.
Die Techniken, die hier eine Rolle spielen sind DOM, HTML, HTTP, CSS, JavaScript, XML, XSLT und der XMLHttpRequest. Da es nicht das Ziel dieser Arbeit ist, auf die Details von Ajax einzugehen, werde ich an dieser Stelle auf den XMLHttpRequest und DOM näher eingehen, da der XMLHttpRequest die wohl wichtigste Technologie für Ajax ist und DOM vor allem für das hier betrachtete LAssi Beispiel von Interesse ist.

10 Ein Browser ist ein Programm, um Internetseiten zu betrachten. Er sorgt für die grafische Darstellung der Inhalte.

DOM

Der Begriff DOM bedeutet Document Object Model und stellt eine Interpretation des Quellcodes eines Dokumentes dar. Im Gegensatz zur „flachen“ Quellcoderepräsentation werden Objekte im DOM über eine Baumstruktur repräsentiert. Dabei werden sowohl die einzelnen Objekte als auch ihre Eigenschaften als Knoten repräsentiert. Dieses wird in folgendem am Beispiel von HTML Quellcode deutlich gemacht. HTML Quellcode besteht aus verschiedenen so genannten Tags, die hierarchisch aufgebaut sind und in spitzen Klammern stehen. So beschreibt das <body> Tag den gesamten Inhaltsteil einer Seite und beinhaltet in der Regel sehr viele andere Tags. Mit Hilfe eines Parsers baut sich der interpretierende Browser eine Repräsentation (ein Modell) der Inhalte des Quelltextes auf. Dabei erzeugt er eine Baumstruktur, die die Zusammenhänge der einzelnen Objekte (Tags) ordnet und in Beziehung setzt. Das oberste Element (die Wurzel) ist dabei das „document“ und alle anderen Bestandteile eines Dokumentes sind über dieses oberste Element (durch Traversieren des Baumes) erreichbar.
Betrachten wir ein kleines Beispiel:

```
(01) <HTML>
(02)    <HEAD>
(03)       <TITLE>Beispielseite</TITLE>
(04)    </HEAD>
(05)    <BODY BGCOLOR="white">
(06)       <H1>Hallo Welt!</H1>
(07)    </BODY>
(08) </HTML>
```

Text 1: Beispiel für eine einfache HTML Seite

Der Text 1 ist ein Beispiel für eine sehr einfache HTML Seite. Sie besteht nur aus einer weißen Seite (durch das Attribut BGCOLOR=“white“ in Zeile 5) und der Überschrift „Hallo Welt!“ (Zeile 6). In der Titelzeile des Browsers steht zusätzlich noch „Beispielseite“ (Zeile 3). Durch die Strukturierung des Codes erkennt man hier schon sehr gut die hierarchische Ordnung, da innerhalb mancher Tags weiter Tags definiert werden. In Zeile 6 sieht man zum Beispiel, dass das Tag <H1>, welches eine große Überschrift repräsentiert innerhalb des <BODY> Tags (Zeile 5 und 7) steht.

In der DOM Baumnotation sieht dieses Dokument so aus wie in Abbildung 3 dargestellt. Das document ist der Wurzelknoten, der nur den HTML Knoten als Kindobjekt referenziert. Der HTML Knoten wiederum hat zwei Söhne, den Head Knoten und den Body Knoten.
So ist jedes Element eindeutig über seine Eltern erreichbar. DOM bietet Darüber hinaus auch Manipulationsmöglichkeiten innerhalb des Baumes, so dass der Nutzer Elemente verändern, hinzufügen oder löschen kann. DOM stellt nicht nur eine Baumstruktur dar, sondern ist vielmehr eine API (Application Programming Interface), die dem User zahlreiche Möglichkeiten bietet, die zu Grunde liegende Struktur zu verändern. Das wird bei Ajax ausgenutzt, um die HTML Seite dynamisch zu verändern, ohne dabei die Seite neu laden zu müssen. In unserem Zeichenprogramm wird DOM später beispielsweise dazu verwendet, um Linien, Kreise, Rechtecke etc. dynamisch in die HTML Seite zu zeichnen.

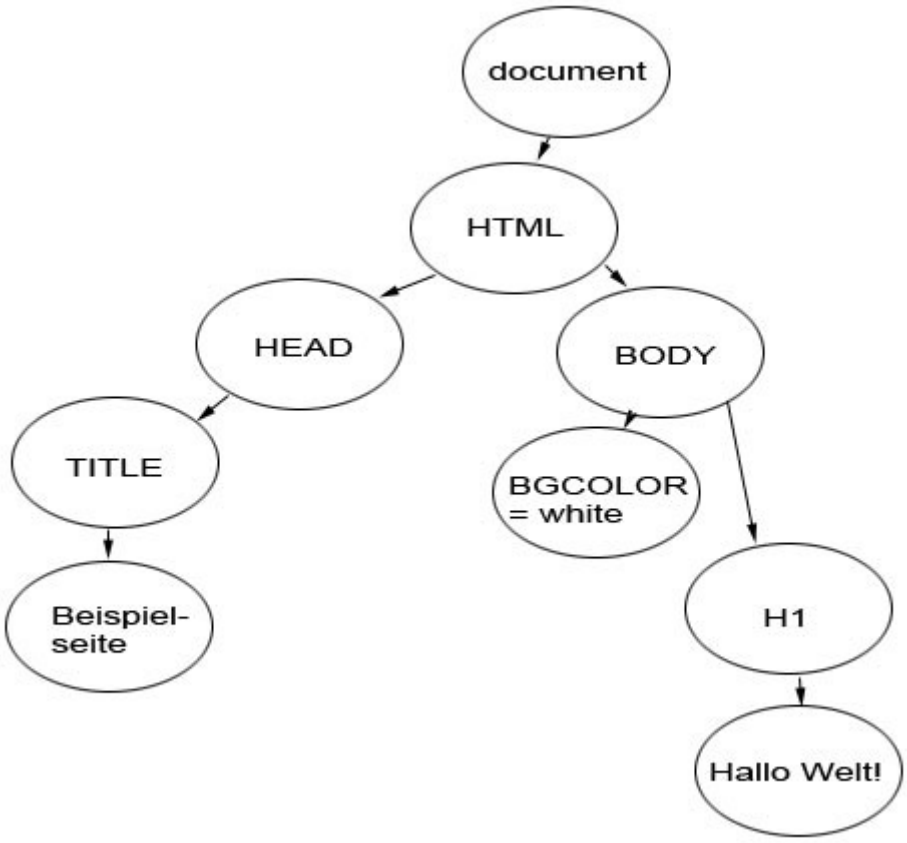


Abbildung 3: Beispiel einer DOM Baumstruktur für Text1

Für eine Übersicht der DOM API siehe [MIN07] Kapitel 2.4.

XMLHttpRequest

Den XMLHttpRequest kann man als die Schnittstelle von JavaScript zum Server bezeichnen, da mit Hilfe dieses Objekttyps Anfragen an den Server gestellt, empfangene Daten verarbeitet und die Statusinformationen gespeichert werden. Damit bildet der XMLHttpRequest „den Kern nahezu jeder AJAX-Anwendung“ [BER05, Kapitel 4.1].
Mit einem XMLHttpRequest kann man sowohl synchrone als auch asynchrone Anfragen durchführen. Um eine Idee davon zu bekommen wie man mit einem XMLHttpRequest umgeht, werde ich beide Möglichkeiten kurz erläutern. Im synchronen Fall wird eine Ressource von einem Server angefordert und das anfordernde Script wartet, bis die Anfrage abgeschlossen ist und bis es eine Antwort erhalten hat.

```
(01) var req = new XMLHttpRequest();
(02) req.open('GET', 'http://www.w3.org/', false);
(03) req.send(null);
(04) handleResponse(req.status, req.responseText);
```

Text 2: Synchroner XMLHttpRequest, aus [BER05, Kapitel 4.1]

Wie im Text 2 zu erkennen ist, wird zunächst ein neuer XMLHttpRequest erzeugt, der eine Anfrage an den Server mit der Adresse „http://www.w3.org“ stellt. Das false in der zweiten Zeile bedeutet dabei, dass die Anfrage nicht asynchron ablaufen soll, so dass das Script nach Absenden des „req.send(null)“ so lange wartet, bis es die Antwort erhält und diese dann mit der Funktion „handleResponse()“ verarbeiten kann.

```
(01) var req = new XMLHttpRequest();
(02) req.onreadystatechange = handleStateChange;
(03) req.open('GET', 'http://www.w3.org/', true);
(04) req.send(null);
```

Text 3: Asynchroner XMLHttpRequest, aus [BER05, Kapitel 4.1]

Im Unterschied dazu kann das Script in Text 3 nach der Anfrage an den Server weiterlaufen und andere Aktionen ausführen. Das ist möglich, da dem Request in Zeile 2 eine so genannte „Callback“-Funktion zugewiesen wird, die immer dann aufgerufen wird, wenn sich der Zustand des Requests ändert. So kann man dem User ermöglichen, Teile der Seite nach und

nach asynchron nachzuladen, so dass er den Eindruck hat, dass ständig etwas passiert.

Ein schönes Beispiel, um die Funktionsweise zu illustrieren, ist dabei Google Maps, welches einen Kartenausschnitt nicht als ein komplettes Bild betrachtet, sondern den Ausschnitt in mehrere kleine Bilder unterteilt. Das hat den Vorteil, dass stets nur einige kleine Bilder nachgeladen werden müssen und der Nutzer sofort sieht, an welcher Stelle sich etwas ändert. Abbildung 4 zeigt einen Ausschnitt von Google Maps, um das Konzept zu illustrieren. Graue Bereiche sind noch nicht geladene Ausschnitte der Karte, die nun asynchron geladen werden. Möchte der Nutzer beispielsweise in dieser Karte einen Schritt nach oben navigieren, so kann ein Großteil der kleinen Bilder beibehalten werden (alle Teilbilder wandern einen Schritt nach oben) und nur ein kleiner Teil von Bilder muss durch das Script nachgeladen werden (in diesem Fall die obere Reihe).

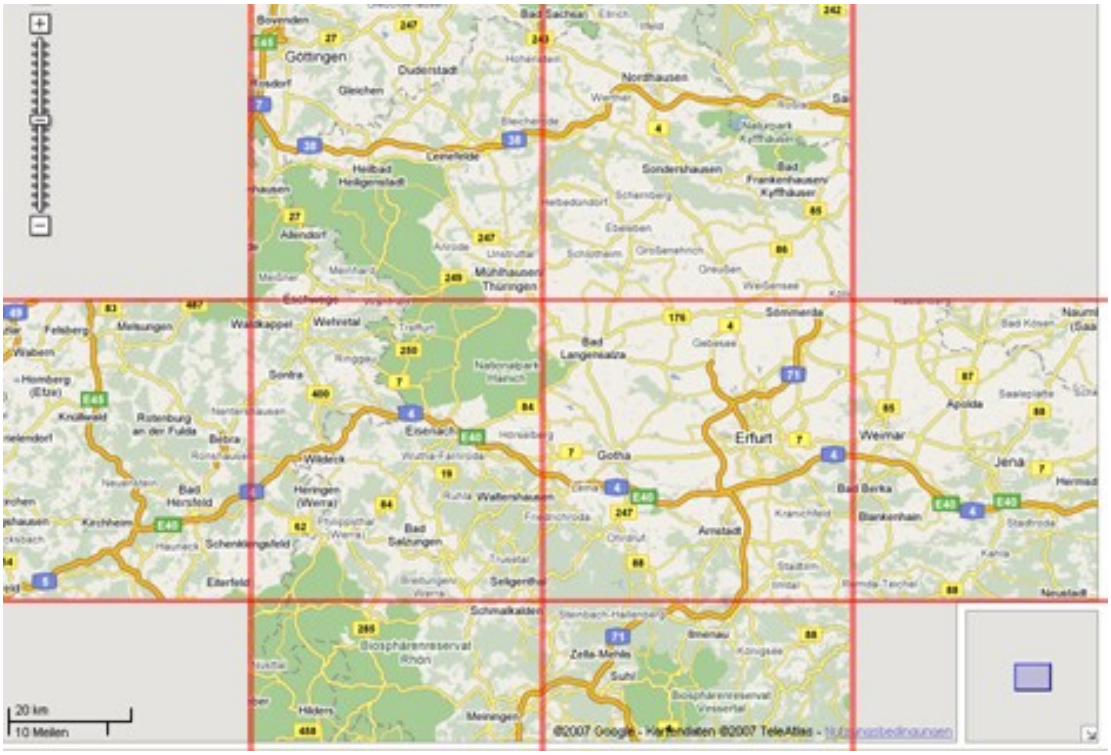


Abbildung 4: Screenshot von Google Maps - rote Linien zur Illustration eingefügt

3) Das Google Web Toolkit (GWT)

Im vorangegangenem Kapitel wurde der Ursprungskontext (die Eclipse Rich Client Plattform) auf der einen und der Zielkontext (eine Webanwendung mit Ajax) auf der anderen Seite vorgestellt. Wie dem Leser vielleicht schon aufgefallen ist, programmiert man im Ursprungskontext auf einem relativ hohen Abstraktionsniveau. Man verwendet die Hochsprache Java, deren Syntax relativ nah am menschlichen Denken orientiert ist, objektorientierte Ansätze unterstützt und die größtenteils maschinenunabhängig ist[11]. Im Gegensatz dazu steht im Zielkontext Ajax die Skriptsprache JavaScript im Vordergrund, die beispielsweise keine strenge Typisierung vornimmt und die vor allem in den unterschiedlichen Browsern unterschiedlich interpretiert wird. Das heißt im schlimmsten Fall für den Entwickler, dass er seine Applikation auf N unterschiedlichen Betriebssystemen in M unterschiedlichen Browsern testen und optimieren muss, so dass es N*M mögliche Kombinationen gibt. Da das offensichtlich ein enormer Aufwand ist, stelle ich in diesem Kapitel das Google Web Toolkit vor, welches den Programmierer von dieser Last befreien kann und welches ich einsetzen werde, um die Migration zu vollziehen.
Das Google Web Toolkit ist eine Open Source Anwendung, die es ermöglicht, Ajax Anwendungen in der Hochsprache Java zu programmieren und die diese dann übersetzt in JavaScript und HTML Dateien, so dass die Anwendung in einem Browser ausgeführt werden kann.
Dabei sorgt das GWT dafür, dass die Anwendung in einer breiten Palette von Browsern funktioniert, da es zwischen den einzelnen Browsern einige Unterschiede gibt, um die man sich selber kümmern müsste, sofern man direkt mit einer Skriptsprache arbeiten würde. Das im vorigen Kapitel angesprochene XMLHttpRequest Objekt wird beispielsweise je nach Browser unterschiedlich verwendet (Vgl. [MIN07] Tabelle S.68).
So erreicht man eine sehr viel höhere Abstraktionsebene und ist in der Lage, seine gewohnte Java IDE verwenden zu können. Damit hat man alle Möglichkeiten des Testens, der Code Vervollständigung und anderer Hilfsmittel, die man aus seiner Entwicklungsumgebung gewohnt ist, als würde man eine „normale“ Java Anwendung realisieren.

11 Auf die Besonderheiten von SWT wird in diesem Kontext nicht näher eingegangen

3.1) Möglichkeiten des GWT

Das GWT besteht aus vier wichtigen Teilen. Der wohl wichtigste wurde dabei bereits in der Einleitung erwähnt und stellt den Java-to-JavaScript Compiler dar, der dafür sorgt, dass der vom Programmierer geschriebene Java Code in JavaScript und HTML Dateien übersetzt wird. Zwei weitere wichtige Teile sind die JRE-Emulationsbibliothek sowie die GW-Web-UI-Klassenbibliothek.

Die JRE-Emulationsbibliothek ermöglicht es, die meistgenutzten Java Klassen auf JavaScript Äquivalente abzubilden. So sind derzeit große Teile des Paketes java.lang und java.util im GWT nutzbar (Für Details, welche Klassen genau und in welchem Umfang nutzbar sind, siehe [GJR07]).

Die GW-Web-UI-Klassenbibliothek stellt Klassen zur Erstellung von Webbrowser-Widgets zur Verfügung. Ein Widget bezeichnet dabei einen eigenständigen Teil der grafischen Benutzungsschnittstelle und ist beispielsweise eine Schaltfläche, ein Eingabefeld oder ein Bild. Es können aber auch komplexe Widgets erzeugt werden, die andere Widgets beinhalten wie etwa ein HorizontalPanel, das mehrere Widgets horizontal nebeneinander anordnet und selber ein Widget ist.

Der vierte Teil betrifft die Möglichkeit des Testens der entwickelten Applikation. Hierfür gibt es im GWT zwei Möglichkeiten:

a) Testen im „Hosted Mode“
   Hier wird die Applikation noch in einer IDE wie Eclipse ausgeführt, das heißt es wird auf Basis des Java Codes getestet und es erfolgt keine explizite Übersetzung in JavaScript Code. Im GWT wird dafür ein integrierter Browser geliefert, der dann die Ausführung zur Laufzeit simuliert.
b) Testen im „Web Mode“
   Hier wird die Applikation in einem Browser nach Wahl getestet. Vorher muss sie daher unter Verwendung des Java-to-JavaScript Compilers in Javascript und HTML Dateien überführt werden. Das ist aber problemlos, denn das ist die Hauptfunktionalität des GWT, so dass der Programmierer überhaupt nichts machen muss, als ein Kompilierscript aufzurufen.

Darüber hinaus bietet das GWT eine Integration zu JUnit Tests und kann mit dem Problem des Zurück Knopfes[12] im Browser zurechtkommen. Weiterhin kann der Entwickler auch eigenen JavaScript Code in die Java Anwendungen integrieren, denn es gibt das so genannte JavaScript Native Interface (JSNI) im GWT (Vgl. [STE07] Kapitel 1.2 und 3.2).

3.2) Erste Schritte mit dem GWT

Dieser Teil meiner Arbeit möchte dem Leser einen kurzen Einblick in die Entwicklung mit Hilfe des GWT geben. Dazu werde ich darauf eingehen, welche Konventionen es im GWT zur Strukturierung eines Projektes gibt und wie man grundsätzlich (unter Windows) vorgeht, um ein neues Projekt in Eclipse anzulegen.

Das GWT ist auf der Seite von Google kostenlos verfügbar ([GWT07]) und kann von dort als gepacktes Verzeichnis heruntergeladen werden. Die Dateien werden in ein beliebiges Verzeichnis entpackt – ich habe das Eclipse Verzeichnis gewählt. Da ein neues Projekt erstellt werden soll, lege ich im nun vorhandenen GWT Verzeichnis ein neues Verzeichnis mit dem Namen „LAssi“ an. Praktischerweise gibt es im Lieferumfang des GWT zwei Skripte, die das Anlegen eines neuen Projektes sehr einfach machen. Das erste Skript lautet projectCreator und dient dem Anlegen eines neuen Projekts. Ich möchte mein Projekt LAssiWebTools nennen und ich möchte das Projekt in Eclipse verwalten, so dass mein Aufruf des Skriptes wie folgt aussieht: „projectCreator.cmd -eclipse LAssiWebTools“.

Das zweite Skript dient dann zum Anlegen des Einstiegspunktes der Anwendung und zum Anlegen der grundlegenden Projektstruktur, wobei darauf zu achten ist, dass der Name beim Aufruf mit „client“ beginnt, da der Einstiegspunkt auf der Clientseite liegt. Der Aufruf lautet bei mir „applicationCreator -eclipse LAssiWebTools client.drawing“, da ich im Projekt LassiWebTools im Client Package eine Klasse drawing anlegen möchte. Zusätzlich erzeugt der applicationCreator die Projektstruktur, indem verschiedene Ordner und Dateien angelegt werden. So wird ein Unterverzeichnis „client“ erzeugt, welches die clientseitigen Java Dateien enthält und ein Unterverzeichnis „server“, das die serverseitigen Dateien enthält. Ein

12 Bei Ajax Anwendungen besteht häufig das Problem, dass der Zurück Knopf im Browser nicht mehr wie vom Benutzer erwartet funktioniert, aber das GWT bietet dem Entwickler Möglichkeiten, das Verhalten des Zurück Knopfes festzulegen.

zusätzliches Unterverzeichnis „public“ enthält die statischen, öffentlich zugänglichen Ressourcen wie HTML Dateien, Bilder und – wie später für unser Zeichenprogramm nötig – eingebettete JavaScript Bibliotheken.

Weiterhin hat das Skript auch die Dateien „drawing-compile.cmd“ und „drawing-shell.cmd“ sowie Eclipse spezifische Informationen in unserem Ordner angelegt. Die „drawing-compile.cmd“ stellt Aufrufe bereit, die dafür sorgen, dass der Java-to-JavaScript Compiler des GWT unser Projekt in JavaScript Dateien übersetzen kann und die „drawing-shell.cmd“ stellt Aufrufe bereit, um unser Projekt im oben besprochenen „Hosted Mode“ auszuführen.
Um den Einstieg zu erleichtern, wird in unserer Einstiegsklasse bereits Code für ein kleines HelloWorld Beispiel erstellt, welches man sofort im „Hosted Mode“ testen kann.
Doch zunächst müssen wir unser Projekt noch in Eclipse integrieren. Dazu wird Eclipse geöffnet und wir wählen im Menü „File“ die Option „Import...“ und dann im sich öffnenden Dialogfenster die Option „Existing Project into Workspace“. Dort wählen wir den soeben erstellten Projektordner und sehen anschließend die von den Skripten erzeugte Packageaufteilung in Eclipse.
Startet man die Anwendung nun im „Hosted Mode“, so ergibt sich das Bild aus Abbildung 5 und durch Drücken Des Knopfes „Click me“ erscheint bzw. verschwindet der Text „HelloWorld!“ je nachdem ob er gerade sichtbar ist oder nicht.

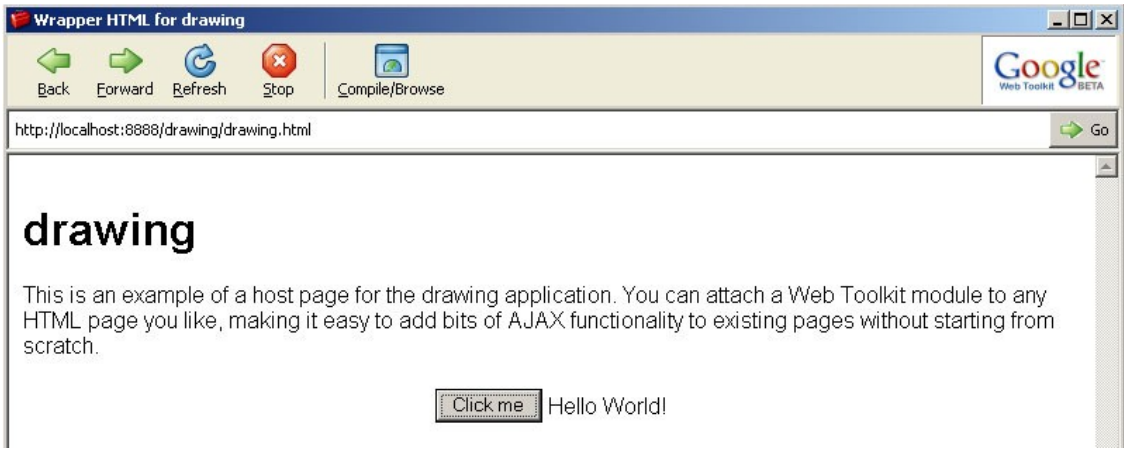


Abbildung 5: Screenshot des "Hosted Modes" nach Erstellung eines neuen Projektes im GWT

Für weitere Informationen zum GWT sei der Leser auf [STE07] und [GWT07] verwiesen.

4) Beschreibung der zu migrierenden Anwendung

In diesem Abschnitt erläutere ich das im Laufe des Projektes entwickelte Zeichenwerkzeug, das dann zu großen Teilen im folgenden Kapitel mit Hilfe des Google Web Toolkits als Webanwendung umgesetzt wird. Ich orientiere mich dabei an dem Stand des Zeichenwerkzeuges, den ich für die Migration benutzt habe und stelle die Funktionen, die noch während der Migration zusätzlich in das Werkzeug geflossen sind, nicht weiter vor. Zunächst wird dazu die Funktionalität des Werkzeuges aus der Anwendersicht beschrieben, um im weiteren Verlauf auch auf die genauere Implementierung einzugehen. Ziel des Kapitels ist es damit, dem Leser den Zusammenhang der einzelnen Packages und Klassen unseres Zeichenwerkzeugs deutlich zu machen. Nicht näher eingehen werde ich auf die Verstrickung des Zeichenwerkzeugs mit LAssi, da dieser Zusammenhang für die Migration nicht näher interessant und relevant ist.

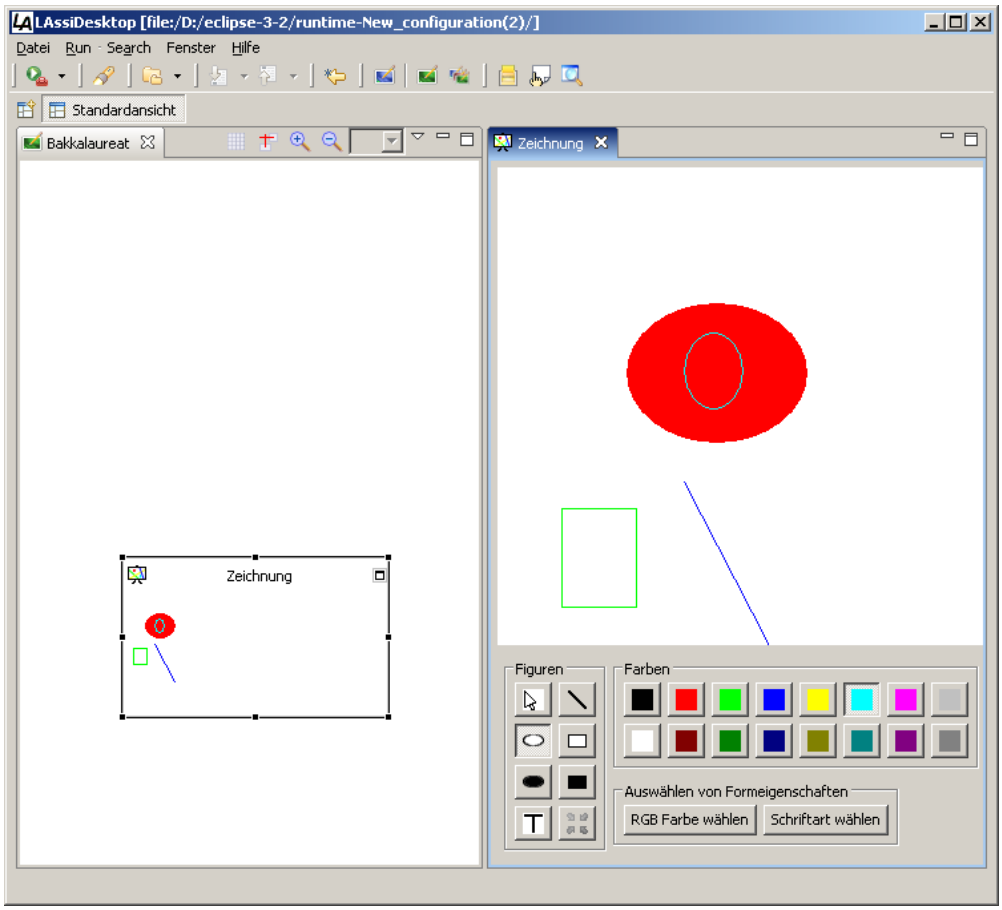


Abbildung 6: Das Zeichenwerkzeug

Das Zeichenwerkzeug wird aktiv, wenn man auf einem Desktop innerhalb von LAssi ein neues Zeichenwerkzeug anlegt. Zeichenwerkzeuge werden wie fast alle Elemente in LAssi als eine Karteikarte repräsentiert. Wenn man diese Karteikarte mit einem Doppelklick aktiviert, so teilt sich das Fenster in zwei Bereiche. Im linken Bereich sieht man weiterhin einen Teil des Desktops (siehe linker Teil der Abbildung 6) unter anderem mit der soeben erstellten Karteikarte für das Zeichenwerkzeug und auf der rechten Seite öffnet sich die Kernfunktionalität der Erweiterung: die Zeichenumgebung (rechter Teil der Abbildung 6). Sie besteht aus einer großen Zeichenfläche, einer Gruppe von Buttons für die verschiedenen Zeichenmethoden, einer Gruppe von Farben und einer Statusanzeige. Beim Öffnen ist standardmäßig das Linienwerkzeug aktiv und als Farbe ist schwarz eingestellt, so dass der Benutzer sofort auf der Zeichenfläche schwarze Linien erschaffen kann. Dazu genügt ein Klick mit der Maus auf einen Punkt (X,Y) der Zeichenfläche bei gleichzeitigem Festhalten der Maustaste. An diesem Punkt befindet man sich in der „Vorschaufunktion“. Das bedeutet, dass nun die Linie verändert wird, je nachdem wie man den Mauszeiger verschiebt. Bewegt man den Mauszeiger zu einem Punkt (X_2,Y_2), so erscheint eine Linie, die die Punkte (X,Y) und (X_2,Y_2) verbindet. Um die Linie wirklich auf die Zeichenfläche zu Zeichnen, lässt der Benutzer die Maustaste wieder los und die Linie ist permanent sichtbar.
Die anderen Werkzeuge arbeiten alle sehr ähnlich und um den Leser nicht zu langweilen, werde ich bei ihnen das Vorgehen weniger detailliert beschreiben, als bei der Linie. Das Rechteckwerkzeug gibt es in zwei Varianten und zwar einmal als gefülltes und einmal als ungefülltes Rechteck. Intuitiv ist klar, dass das eine Werkzeug nur den Rahmen eines Rechtecks ins Bild zeichnet wohingegen das andere ein vollständig ausgefülltes Rechteck erstellt. Analog zum Rechteckwerkzeug gibt es ein Werkzeug für gefüllte und eines für ungefüllte Ellipsen.
Etwas anders verhalten sich die verbliebenen Werkzeuge. Das Auswahlwerkzeug (symbolisiert durch einen gefüllten Pfeil) ermöglicht es, auf der Zeichenfläche gezeichnete Objekte auszuwählen und zu verschieben. Man kann ein einzelnes Objekt anklicken und direkt durch gedrückt halten der Maustaste verschieben, oder man kann ein Rechteck aufziehen (wie beim Rechteckwerkzeug), das alle Figuren seiner Fläche automatisch markiert. Ausgewählte Objekte werden durch einen gestrichelten Rahmen hervorgehoben, um dem Benutzer klar zumachen, welche Objekte er gerade im Visier hat.
Schließlich gibt es noch das Werkzeug zum Gruppieren von Objekten. Wenn Objekte

gruppiert sind, bedeutet das, dass sie wie ein einzelnes Objekt behandelt werden, dass sie also nur alle gemeinsam markiert und verschoben werden können. Der Benutzer gruppiert Objekte, indem er die zu gruppierenden Objekte mit Hilfe des Auswahlwerkzeuges markiert und dann einmal auf den Gruppieren Knopf drückt. Es ist ebenfalls möglich, hierarchische Gruppen zu bilden. Angenommen, ich habe verschiedene Objekte A,B und C zu einer Gruppe G gruppiert und andere Objekte D,E und F zu einer Gruppe H gruppiert. Dann kann ich die Gruppe G und die Gruppe H jeweils wieder als Objekte betrachten, die ich in einer Gruppe I vereinige, so dass ich am Ende nur ein großes Objekt habe, welches hierachisch andere Objekte enthält.

Nach der Vorstellung dieser Erweiterung für LAssi, wird es nun im weiteren Verlauf des Kapitels um die Implementation des Zeichenwerkzeugs gehen. Dabei soll es nicht so sehr um den Prozess der Entwicklung, um die verschiedenen Herausforderungen bei der Entwicklung sowie um die Vor- und die Nachteile des Entwurfes gehen, sondern vielmehr um den Aufbau und die Interaktion der verschiedenen Teile des Werkzeuges, um dem Leser einen Einblick in die inneren Komponenten zu gewähren, so dass er etwas besser einschätzen kann, welcher Teil dieser bestehenden Implementation bei der Migration tatsächlich produktiv wieder verwertet werden konnte. Schließlich ist der interessanteste Aspekt dieser Arbeit für den Leser, inwieweit eine angestrebte Migration realisierbar ist und wie viel des Codes man dabei neu schreiben muss.

Das vorliegende Zeichenwerkzeug besteht aus den Plugins „org.lassitools.drawing“ und „org.lassitools.drawing.ui“. Ersteres ist dabei für die fachliche Logik und den Persistenzdienst zuständig und letzteres für die grafische Repräsentation der Anwendung.
Der Persistenzdienst sorgt dafür, dass gemalte Zeichnungen gespeichert werden, so dass nach dem Beenden und Neustarten von LAssi die erstellten Zeichnungen nicht verlorengehen. Der Persistenzdienst ist an dieser Stelle jedoch weniger interessant, da die Funktionalität zunächst nicht bei der Migration betrachtet wird.
Die fachliche Logik besteht aus Klassen, die für die einzelnen Figuren wie Rechteck, Ellipse und Linie stehen. Sie implementieren dabei alle das Interface IFigure, so dass sie einheitlich behandelt werden können. Auf dem Interface IFigure wiederum setzt die abstrakte Klasse AbstractFigure auf, die alle den Figuren gemeinsamen Operationen implementiert wie beispielsweise verschiedene Getter- und Settermethoden zum Setzen und Abrufen von

Figureigenschaften. So erbt jede Figurenklasse von der AbstractFigure Klasse und muss selber nur einige wenige Operationen implementieren wie zum Beispiel die Methode draw(GC gc), die die Figur auf dem GC (Graphics Context) anzeigt oder die Methode contains(int x, int y), die true oder false liefert, je nachdem, ob ein Teil der Figur am Punkt (x,y) vorkommt.
Kern des Plugins für die grafische Repräsentation ist die Klasse DrawingEditor. Hier liegt zugleich auch der Kern der gesamten Anwendung, da der DrawingEditor sowohl die Zeichenfläche, als auch die Knöpfe zur Steuerung des Programms verwaltet. Daher werden auch die verschiedenen Maus- und Tastaturlistener im DrawingEditor an ihre Komponenten, wie etwa den Knöpfen zur Auswahl der Funktionen und der Zeichenfläche für die Mauszeichenoperationen, gehängt.
Die Zeichenfläche selbst ist dabei in eine eigene Klasse mit dem Namen DrawingCanvas ausgelagert, die vom DrawingEditor verwendet wird. Der DrawingCanvas beinhaltet einen PaintListener, der gegebenenfalls alle vorhandenen Figuren auf die Zeichenfläche zeichnet bzw. der an den entsprechenden Figuren ihre draw(GC gc) Methoden aufruft.

5) Vorgehen

An dieser Stelle meiner Arbeit habe ich alle nötigen Voraussetzungen für den Leser gelegt, um auf die erfolgte Migration der Anwendung von der Eclipse Rich Client Plattform (siehe Kapitel 2.2) in das Google Web Toolkit (siehe Kapitel 3) einzugehen.
Primäres Ziel der Migration war dabei, die Menge des vorhandenen Codes möglichst in ihrer Gesamtfunktionalität zu erhalten. Da das Zeichenwerkzeug aus einer größeren Menge von Code besteht, habe ich mich dazu entschlossen, das Vorgehen in mehrere Schritte zu trennen. In einem ersten Schritt habe ich daher die grafische Benutzungsschnittstelle (GUI[13]) entwickelt, um mit der ersten „Version“ zumindest eine äußere Ähnlichkeit zu erreichen. Nachdem diese Version einwandfrei funktionierte, bin ich dazu übergegangen, einen sehr kleinen Teil der Funktionalität[14] des Programms tatsächlich abzubilden.
Als auch dieser Teil verwirklicht war, konnte ich sukzessive auch die anderen Werkzeuge in die GWT Klassen portieren und so fast die gesamte Funktionalität der Anwendung abbilden. Dieses Kapitel wird nun die einzelnen Schritte näher beschreiben und auf dabei entstandene Probleme und deren Lösungen eingehen.

5.1) Erster Schritt: Grafische Benutzungsschnittstelle

Bei der Portierung der grafischen Benutzungsschnittstelle musste ich am stärksten von der zu Grunde liegenden Anwendung abstrahieren, da unser Zeichenwerkzeug in den Kontext von LAssi eingebettet ist und ich dieses aus dem Kontext gelöst habe. So ist es in der Ursprungsanwendung so, dass man für Zeichenflächen Karteikarten erstellen kann, die bei einem Doppelklick mit der Maus dann eine Zeichenfläche öffnen (Vgl. Kapitel 4). Aus zeitlichen Gründen[15] habe ich mich dazu entschlossen, nur die unmittelbare Zeichenfläche zu modellieren, so dass die Repräsentation als Karteikarte sowie der restliche LAssi Kontext nicht mit modelliert wurde.

13 Graphical User Interface

14 hier: lediglich das Rechteck-Werkzeug unseres Programms

15 Zeitliche Gründe meint hierbei zum einen die begrenzte für die Arbeit zur Verfügung stehende Zeit und zum anderen zusätzliche Zeit, die nötig gewesen wäre, um mich in den anderen (nicht von meiner Gruppe) entwickelten Quellcode einzuarbeiten.

So kam der grafischen Benutzungsschnittstelle bei der Migration eine leichte Sonderrolle zu, da es mir hier nicht praktikabel erschien, den Code unserer Anwendung als Grundlage zu nehmen. Stattdessen habe ich die grafische Benutzungsschnittstelle zunächst relativ losgelöst vom Quellcode unseres Werkzeugs im Zuge der Einarbeitung in das Google Web Toolkit erstellt. Dieser erste Schritt war damit einer der schwierigsten Teile der Migration, da die grafischen Elemente im GWT sich grundlegend von den grafischen Elementen in Eclipse unterscheiden, da sie letztlich auf im Browser darstellbare Elemente abbildbar sein müssen. Im Hinblick auf das weitere Vorgehen habe ich gleich darauf geachtet, dass die verwendeten Elemente auch für die folgenden Schritte verwendbar sein würden. Das bedeutet zum Beispiel, dass die Zeichenfläche die Möglichkeit bieten musste, Maus Events[16] zu erkennen und zu verarbeiten. Außerdem sollte sie natürlich auch die Möglichkeit bieten, darauf zu Zeichnen, so dass der erste und der zweite Schritt nicht wirklich vollständig getrennt waren, sondern ich mir schon zu diesem Zeitpunkt die verschiedenen vorhandenen Möglichkeiten angesehen habe. Aus Gründen der besseren Konzeption beschreibe ich alles zum Zeichnen aber geschlossen im zweiten Schritt.

Neben der Zeichenfläche war die zweite wichtige Komponente der grafischen Benutzungsschnittstelle die Knöpfe, um zum Beispiel die Farbauswahl zu steuern. Diese lagen in der Eclipse Anwendung als so genannte Toggle Knöpfe vor. Toggle bedeutet dabei, dass ein Knopf entweder den Zustand „an“ oder „aus“ hat. Wenn er „an“ ist, wird das grafisch beispielsweise durch Schattierung hervorgehoben. Solche Knöpfe wollte ich nun auch im Browser darstellen können, aber GWT selber bietet solche Knöpfe nicht an. In der GWT Widget Library [WID07] gab es allerdings eine Klasse ToggleButton. Leider funktionierte diese nicht so wie erwünscht, denn sie hatte einen logischen Fehler[17] im Quellcode. Ich habe den zuständigen Autor der Klasse auf den Fehler hingewiesen und er hat mir auch versichert, dass der Fehler im nächsten Release gefixt würde. Um nicht auf das nächste Release warten zu müssen, habe ich eine eigene Klasse ToggleButtonEdit programmiert, die genau wie die ToggleButton Klasse operiert, die aber den Fehler bereits korrigiert.
Der Listener für die ToggleButtons ist ebenfalls etwas anders als im SWT, denn im SWT

16 Maus Events sind beispielsweise das Bewegen des Mauszeigers auf der Zeichenfläche, das Drücken einer Maustaste auf der Zeichenfläche sowie das Loslassen einer Maustaste auf der Zeichenfläche

17 Der Fehler führte dazu, dass die Knöpfe noch markiert waren, obwohl sie wieder deaktiviert wurden

verwendet man einen SelectionListener und im GWT einen ClickListener. Abgesehen von Umbenennungen ist die Anpassung für unser Zeichentool aber unproblematisch.
Um die verschiedenen Komponenten nun auf der HTML Seite zu platzieren, benutze ich die vom GWT bereitgestellte Klasse FlexTable. Eine FlexTable ist eine Tabelle mit beliebig vielen Spalten und Zeilen mit der Besonderheit, dass man auch verschiedene Zellen verschmelzen kann. So ist es möglich, seine Komponenten flexibel relativ zueinander anzuordnen. Im Falle des Zeichenprogramms habe ich die FlexTable wie in Abbildung 7 gezeigt aufgebaut.

Zeichenfläche	
WerkzeugButtons	*FarbButtons*

Abbildung 7: Aufbau des Layouts mit einer FlexTable

Die Tabelle besteht aus zwei Zeilen bei der in die erste Zeile die Zeichenfläche eingefügt wird (die beiden Zellen der Zeile wurden also zu einer Zelle verbunden). In der zweiten Zeile werden in die linke Zelle die Werkzeug Buttons und in die rechte Zelle die Farb Buttons eingefügt. Um die Buttons jeweils zu einem Widget zu vereinen, benutze ich dafür ein so genanntes HorizontalPanel, welches mehrere Elemente horizontal nebeneinander anordnet.
Weniger problematisch war die Umstellung der Farben, da die Klasse Color aus SWT nicht im Web Toolkit von Google unterstützt wird. Es gibt nun im Interface IFigure an Stelle der bisherigen Definition von RGB Werten der Form

```
static final RGB BLACK = new RGB(0, 0, 0);
```

Text 4: Defintion eines RGB Wertes im Interface IFigure vorher

eine direkte Definition der Color Klasse der GWT Widget Library:

```
static final Color BLACK = new Color(0, 0, 0);
```

Text 5: Defintion eines Color Wertes im Interface IFigure nachher

5.2) Zweiter Schritt: Funktionalität des Rechteckwerkzeuges

An dieser Stelle war es Zeit, die implementierte grafische Oberfläche mit Leben zu füllen. Um dieses zu realisieren, habe ich die Klassenstruktur aus unserem Zeichenwerkzeug komplett übernommen und zunächst fast alles auskommentiert, um nicht mit sehr vielen Fehlern überflutet zu werden. Das Ziel zu dieser Zeit, war es, eine erste Grundfunktionalität zu implementieren, um die Gewissheit zu haben, dass das geplante Vorhaben überhaupt realisierbar ist. Zu diesem Zwecke habe ich mit der einfachen Funktion des Zeichnens eines Rechtecks begonnen.
Hier trat aber bereits das erste größere Problem auf, denn es gibt keine Klassen im GWT, die das Zeichnen von Rechtecken, Kreisen, Linien und ähnlichen Zeichenoperationen implementieren, so dass ich mich umgesehen habe, was es noch für Erweiterungen zum GWT gibt.
Dabei habe ich mich auf der Seite [BUR06] über vorhandene Widgets Erweiterungen für das GWT erkundigt und bin dabei auf die GWT Widget Library [WID07] gestoßen, da diese einen Wrapper[18] für das Paket JsGraphicsPanel, das von Walter Zorn entwickelt wurde ([ZOR06]), um Zeichenwerkzeuge browserübergreifend darzustellen, bereitstellt.
Die Bibliothek von Walter Zorn benutzt eingefärbte -Layer, um im Browser die verschiedenen Zeichenfiguren darzustellen. Wenn nun für jeden Pixel einer Linie ein eigener Layer erstellt wird, ist das nicht besonders effizient. Daher versucht das Paket von Walter Zorn, gleichfarbige, nebeneinander liegende Pixel zu einem Layer zusammenzufassen (Vgl. Abbildung 8).

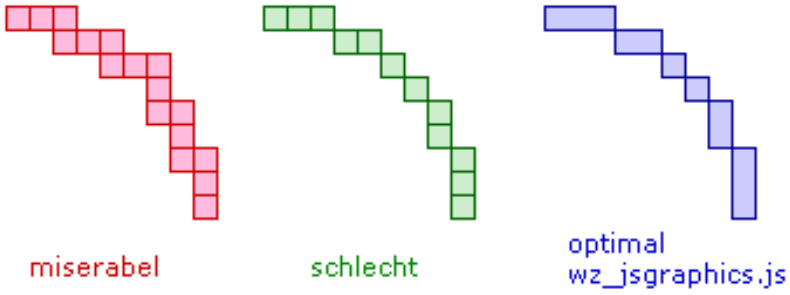
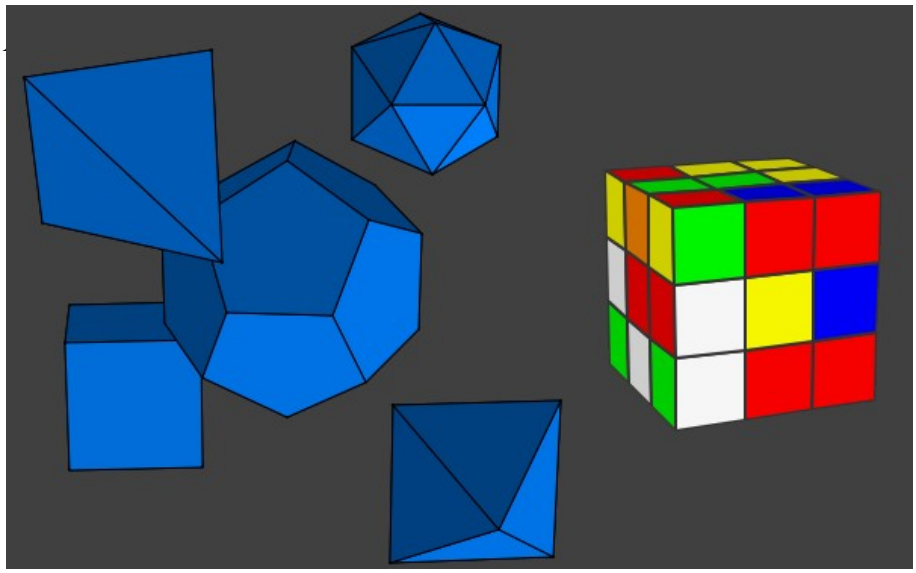


Abbildung 8: Grafische Darstellung eines Viertelkreises durch farbige -Layer

18 Wrapper bedeutet hier, das die Widget Library eine Klasse anbietet, die die Funktionalität des JavaScript Paketes abbildet und damit dieses auf Java Basis kapselt.

Trotzdem sagt selbst der Autor, dass die Performanz nicht mit einem Java Programm vergleichbar sein kann und dass diese Bibliothek nur der Versuch ist, so gut wie möglich mit den webtechnologisch bedingten Einschränkungen zurecht zu kommen ([ZOR06]).
Aus diesem Grunde, habe ich mich noch weiter umgesehen und habe im Laufe der Zeit die skalierbaren Vektorgrafiken (SVG) entdeckt, die mir anfangs sehr viel versprechend erschienen. Skalierbare Vektorgrafiken beschreiben Vektorgrafiken in der XML Syntax und wurden im September 2001 vom W3C[19] als Empfehlung veröffentlicht. Große Teile von SVG laufen auch in den gängigen Browsern (mit Plugin im Internet Explorer) und es gibt bereits einige vielversprechende Anwendungen wie beispielsweise Lutz Tautenhahns SVG-VML-3D. Das ist eine JavaScript Bibliothek, um dreidimensionale Grafiken wie komplexe Diagramme mit JavaScript darzustellen. Es liefert beachtliche Ergebnisse wie etwa in Abbildung 9 dargestellt.
Leider habe ich keine Bibliothek gefunden, die eine ähnliche Funktionalität für zweidimensionale Bilder in JavaScript kapselt und die Einbindung von SVG im Google Web Toolkit ist auch nur marginal unterstützt (in der GWT Widget Library). Ab Version 1.4 des GWT werden skalierbare Vektorgrafiken sogar überhaupt nicht mehr unterstützt, da die Version 1.4 die bisherige Einbindung von SVG durch die GWT Widget Library nicht mehr ermöglicht (Vgl. [HAN07] Kommentar vom 15.05.2007).
Aus diesen Gründen habe ich mich dazu entschlossen, die Bibliothek von Walter Zorn (siehe oben) zu verwenden, da diese zuverlässig funktioniert und gut ins GWT eingebunden werden kann.



19 Das World Wide Web Consortium ist ein Gremium zur Standardisierung von Web Techniken

Jegliche Verwendung der Klasse GC (Graphics Context) in unserem Projekt musste nun also durch eine Verwendung der Klasse JsGraphicsPanel ersetzt werden. Die beiden Klassen sind aber nicht direkt substituierbar, da es einige kleinere Unterschiede gibt. So unterstützt die GC Klasse das Setzen von Vorder- und Hintergrundfarbe für ihre Zeichenoperationen wohingegen die Klasse JsGraphicsPanel mit der zu Grunde liegenden JavaScript Bibliothek nur eine Farbe unterstützt, die gleichzeitig Vorder- und Hintergrundfarbe ist (Vgl. Text 6 Zeile 4 und 5 mit Text 7 Zeile 3 und 4). Das ist für die Anwendung bisher kein großes Problem, es in der Architektur zwar eine solche Trennung in Vorder- und Hintergrundfarbe gibt, sie aber noch nicht funktional unterstützt wird und es faktisch immer nur eine Farbe zur Zeit gibt.
Etwas größeren Aufwand verursachte dagegen der Unterschied beim Zeichnen eines Rechtecks, da die Methode drawRectangle der GC Klasse negative Breite und Höhe eines Rechtecks als Eingabe akzeptiert, die Methode drawRect der JsGraphicsPanel Klasse aber ausschließlich positive Werte der Breite und Höhe annimmt.

```
(01)  int width = getEndX() - getStartX();
(02)  int height = getEndY() - getStartY();
(03)
(04)  gc.setForeground(getBorderColor());
(05)  gc.setBackground(getFillColor());
(06)  // Hintergrund ggf. malen
(07)  if(isFilled()){
(08)        gc.fillRectangle(getStartX(),
(09)              getStartY(), width, height);
(10)  }
      //... Teile des Codes ausgelassen
(11)  gc.drawRectangle(getStartX() + a, getStartY() + b, width - 2
            * a, height - 2 * b);
(12)  // Falls die Figur selektiert ist, einen Auswahlrahmen malen
(13)  if(isSelected()){
(14)        gc.setLineStyle(SWT.LINE_DOT);
(15)        gc.setForeground(new Color(Display.getCurrent(),
                  IFigure.BLACK));
(16)        //Abstand des Auwahlrahmens vom Rechteck in Pixeln
(17)        final int abstand = 2;
(18)        gc.drawRectangle(getStartX() - abstand, getStartY()
                        -  abstand, width + abstand + abstand,height +
                           abstand + abstand);
(19)        gc.setLineStyle(SWT.LINE_SOLID);
```

Text 6: Auszug der draw Methode der RectangleFigure Klasse vorher

Um dieses Problem zu beheben, wird vor dem Aufruf der Methode drawRect() geprüft, ob die Höhe oder die Breite negativ ist. Falls die Breite negativ ist, so wird der X-Wert des Startpunktes des Rechtecks um diese negative Breite nach links verschoben und die Breite

wird durch eine Multiplikation mit minus eins positiv gemacht (Vgl. Text 7 Zeile 9-14). So erreicht man, dass ein Rechteck mit positiver Breite gezeichnet wird. Analog wird im Falle einer negativen Höhe der Y-Wert des Startpunktes um die negative Höhe nach oben verschoben und die Höhe wird anschließend mit minus eins multipliziert, so dass auch die Höhe positiv wird (Vgl. Text 7 Zeile 15-20). Offensichtlich ändert sich an dem gezeichneten Rechteck nichts, da die Koordinaten nur gegenseitig verrechnet werden, denn nehmen wir als Beispiel für den X-Wert 100 und für die Breite -30 (Das entspricht einer Linie vom X-Wert 100 um 30 Pixel nach links zum X-Wert 70). Dann wird durch die Umrechnung der X-Wert zu 70 und die Breite zu 30, so dass das einer Linie vom X-Wert 70 nach rechts zum X-Wert 100 entspricht, was die gleiche Linie beschreibt.
Ebenfalls etwas anders ist es in der Klasse JsGraphicsPanel die Art der Linienbeschaffenheit (LineStyle) einzustellen, denn es gibt zwar eine Methode setStrokeDotted(), aber keine Methode setStrokeNormal() oder ähnliches, aber eine Lösung ist hier einfach die Methode setStrokeWidth(1) zu rufen, die die Linienbeschaffenheit wieder auf solid setzt und die Linienstärke auf 1 setzt (Vgl. Text 6 Zeile 14 und 19 mit Text 7 Zeile 28 und 32).

```
(01)  int width = getEndX() - getStartX();
(02)  int height = getEndY() - getStartY();

(03)  //JsGraphicsPanel unterstützt nur eine Farbe zur Zeit
(04)  jsgp.setColor(getBorderColor());

(05)  //Workaround, JS akzeptiert keine neg. Werte für width und height,
(06)  //also muss man an dieser Stelle Koordinaten umrechnen!
(07)  int startX = getStartX();
(08)  int startY = getStartY();
(09)  if (width < 0) {
(10)        //X wird nach links verschoben
(11)        startX = startX + width;
(12)        //Breite positiv
(13)        width = width * (-1);
(14)  }
(15)  if (height < 0) {
(16)        //Y wird nach oben verschoben
(17)        startY = startY + height;
(18)        //Höhe positiv machen
(19)        height = height * (-1);
(20)  }

(21)  // Hintergrund ggf. malen
(22)  if(isFilled()){
(23)        jsgp.fillRect(startX, startY, width, height);
(24)  }

(25)  jsgp.drawRect(startX, startY, width, height);
(26)  // Falls die Figur selektiert ist, einen Auswahlrahmen malen
(27)  if(isSelected()){
(28)        jsgp.setStrokeDotted();
(29)        jsgp.setColor(Color.BLACK);
(30)        final int abstand = 2;
(31)        jsgp.drawRect(getStartX() - abstand, getStartY()
               -   abstand, width + abstand + abstand,
                        height + abstand + abstand);
(32)        jsgp.setStrokeWidth(1);
(33)  }
```

Text 7: Auszug der draw Methode der RectangleFigure Klasse nachher

Ähnlich wie bei den weiter oben beschriebenen Listenern für die ToggleButtons, sind auch die Listener für die Mausevents im GWT etwas anders als im SWT. So benötigt man im SWT einen MouseListener für die Klick Events und einen MouseMoveListener für die Erfassung der kontinuierlichen Bewegung der Maus, wohingegen im GWT alle Methoden gemeinsam im MouseListener vereinigt werden, so dass die Hauptarbeit auch hier in der Umbenennung der zugehörigen Klassen liegt. Zusätzlich werden den MouseListenern im GWT nicht wie in SWT MouseEvents übergeben, die die gesamte Information wie die Position der Maus etc.

beinhalten, sondern ein Tripel aus dem Widget in dem das Event stattgefunden hat sowie dem X- und dem Y-Wert der aktuellen Mausposition. Aber auch diesen Unterschied kann man durch minimale Änderungen ausgleichen, da überall nur die Verwendung von Event.X bzw. Event.Y durch die direkte Verwendung von X bzw. Y ersetzt werden muss.
Nachdem alle Änderungen erfolgten und die Eclipse IDE keine Konflikte mehr angezeigt hat, konnte der erste Funktionstest durchgeführt werden. Dieser wurde aber sehr schnell durch die Fehlermeldung „[ERROR] Line 885: GWT does not yet support the Java 5.0 language enhancements; only 1.4 compatible source may be used“ abgebrochen, so dass weitere Anpassungen nötig waren, da das GWT offensichtlich nur den Java Sprachstandard 1.4 und nicht die im Sprachstandard 5.0 verfügbaren neuen Konstrukte unterstützt.
Für die Umstellung auf Java 1.4 waren im wesentlichen drei Arten von Änderungen zu unterscheiden. Die erste Art ist die Verwendung der neuen for Schleife, die es in Java 1.4 noch nicht gibt. Ein Beispiel für ein solches Konstrukt ist in Text 8 gezeigt. Hier wird die Variable figure vom Typ IFigure automatisch mit dem jeweils nächsten Wert aus der Liste von _drawing.getFigures() belegt (Zeile 1) und kann dann im Rumpf der Schleife benutzt werden, um die jeweilige Figur zu deselektieren (Zeile 2).

```
(01) for(IFigure figure : _drawing.getFigures()) {
(02)    figure.setSelected(false);
(03) }
```

Text 8: Beispiel einer „neuen“ for Schleife

In Java 1.4 muss man das ein wenig umständlicher realisieren, indem man sich zunächst von der zu iterierenden Liste einen so genannten Iterator holt, der dann über die Methode next() das jeweils nächste Element der Liste liefert. Hier muss man zusätzlich auch immer einen manuellen Cast auf das Interface IFigure durchführen, da Java 1.4 noch keine Generics unterstützt (siehe unten). Eine Möglichkeit, die obige Schleife in den 1.4 Standard zu überführen, zeigt der Text 9.

```
    (01) Iterator it = _drawing.getFigures().iterator();
(02) while (it.hasNext()) {
(03)    IFigure figure = (IFigure) it.next();
(04)    figure.setSelected(false);
(05) }
```

Text 9: Beispiel der Umwandlung der „neuen“ for Schleife nach Java 1.4

Eng verbunden damit ist die Umstellung der Generics. Generics sind ein Konstrukt von Java 5.0, das es Listen und Maps ermöglicht, zu definieren welche Art von Elementen in ihnen vorhanden sind. So kann man Fehler bereits im Vorwege vermeiden, wenn sicher ist, dass nur Elemente eines Typs in einer Liste vorkommen. In Java 1.4 ist das nicht möglich. Stattdessen können in einer Liste beliebige Objekte vorkommen, die man, sofern man eine speziellere Klasse benötigt, „casten“ muss (siehe Text 9 Zeile 3).
Die dritte Umstellung betrifft Enums. Ein Enum ist eine geordnete Aufzählung von Elementen, die intern automatisch nummeriert werden wie beispielsweise in Text 10.

```
(01) private enum Figures {
(02)    LINE, OVAL, RECTANGLE, OVAL_FILLED, RECTANGLE_FILLED, SELECTION
(03) }
```

Text 10: Beispiel eines Enum

Da es in Java 1.4 noch keine Enums gibt, müssen solche Konstrukte anders repräsentiert werden. Ich habe mich dafür entschieden, daraus eine private Klasse zu machen, die die einzelnen Figuren als statische Integer-Konstanten beinhaltet, so dass die Umsetzung wie in Text 11 erfolgte.

```
(01) private final static class Figures {
(02) private final static int LINE = 0;
(03) private final static int OVAL = 1;
(04) private final static int RECTANGLE = 2;
(05) private final static int OVAL_FILLED = 3;
(06) private final static int RECTANGLE_FILLED = 4;
(07) private final static int SELECTION = 5;
(08) }
```

Text 11: Beispiel einer privaten Klasse als Ersatz eines Enums

Nach der Anpassung an den Java 1.4 Standard war der erste Funktionstest erfolgreich. Allerdings funktioniert das Programm überraschenderweise im Internet Explorer nicht ordnungsgemäß, obwohl einer der großen Vorteile des GWT sein soll, den Programmierer von browserspezifischen Unterschieden abzuschotten. Der Internet Explorer scheint Probleme mit der Umsetzung des MouseMove Listeners zu haben, denn ohne diesen funktioniert es auch im Internet Explorer.

5.3) Dritter Schritt: Sukzessive Erweiterung

Nachdem im ersten Schritt die grafische Benutzungsoberfläche erstellt wurde und im zweiten Schritt das Rechteckwerkzeug implementiert wurde, galt es nun im dritten Schritt, den bisher zu sehr großen Teilen auskommentierten Quellcode Stück für Stück mit ins Zeichenwerkzeug im GWT einzubinden.
Die Trennung unterschiedlicher Zeichenfiguren in unserer Architektur machte dieses Vorgehen sehr komfortabel, da ich so eine Figur nach der anderen einbinden konnte. Zur eigentlichen Einbindung ist zu sagen, dass das Vorgehen dem des Rechteckwerkzeuges sehr nahe kam. So habe ich zunächst stets die Auskommentierung des Quellcodes entfernt, dann die gesamten Umbenennungen der zum GWT nicht kompatiblen Klassen vorgenommen (unter anderem die Klassen Color, GC, MouseListener, ClickListener) und nicht benötigte SWT Klassen wie beispielsweise Display entfernt.
Anschließend habe ich die Inkompatibilitäten bezüglich der Java Version ausgeglichen und den Quellcode auf den Java 1.4 Standard gebracht (siehe zweiter Schritt). Die einzige größere Anpassung gab es dann stets in der paint() Methode der jeweiligen Figur, da hier wie besprochen nun an Stelle des GC das JSGraphicsPanel eingesetzt wird und man die Funktionalität leicht anpassen musste.
Eine Besonderheit gab es noch bei der Klasse OvalFigure, die für das Zeichnen von Ellipsen zuständig ist, da diese in der Methode contains() auf die Klasse Ellipse2D des AWT zugegriffen hat und sich eine ähnliche Klasse im GWT nicht finden lässt, so dass ich diese Methode selber implementieren musste.

6) Erfahrungen

An dieser Stelle meiner Arbeit möchte ich über die im Laufe des Prozesses gemachten Erfahrungen der Migration einer Rich Client Anwendung in eine Webanwendung unter Verwendung des GWT berichten.
Um die gemachten Erfahrungen besser einordnen zu können, gehe ich zunächst kurz auf die für mich positiven und negativen Voraussetzungen ein, damit sich der geneigte Leser ein Bild machen kann, ob eine etwaige Migrationsabsicht bei ihm unter ähnlichen Voraussetzungen stattfindet und daher vergleichbar ist, oder ob es grundlegende Unterschiede gibt.
Als für mich positive Voraussetzung erachte ich die sehr gute Kenntnis des zu Grunde liegenden Quellcodes, da ich an der Programmierung von Anfang und in einem Zeitraum eines Jahres selber mitgewirkt habe. Weiterhin handelt es sich noch im Vergleich zu manch anderem Projekt um eine übersichtliche Anzahl von Codezeilen, so dass ich den gesamten Zusammenhang der Klassen im Vorwege kannte. Weiterhin positiv war die lose Kopplung unseres Zeichentools an die LAssi Plattform, so dass ich dieses ohne großartige Probleme aus dem Kontext lösen konnte.
Ein weiterer Vorteil ist die sehr gute Strukturierung des ursprünglichen Quellcodes, da dieser Quellcode in einem Universitätsprojekt entstanden ist in dem es ausreichend Zeit für die Verwendung moderner Entwicklungsmethoden wie testgetriebene Entwicklung, Pair Programming und das Vertragsmodell gab, so dass man die Qualität des Codes als sehr gut bewerten kann. Hinzu kommt dabei, dass das Zeichenprogramm selber aus ähnlich strukturierten Elementen (den Figuren) besteht, so dass ein schrittweises Vorgehen hier möglich war.
Trotzdem kann man die hohe Qualität des Codes gleichzeitig auch als eine negative Voraussetzung zählen, da dieser nach dem Java 5.0 Standard programmiert wurde und das GWT derzeit nur den 1.4 Standard unterstützt. So musste ich zahlreiche Sprachkonstrukte wie enums, for Schleifen und Generics (siehe auch Kapitel 5) umstellen, was zusätzliche Arbeit bereitet hat.
Ein weiterer zentraler Nachteil ist aus meiner Sicht die Art der Anwendung, da gerade die aufwendigen Zeichenoperation, die man bei unserem Zeichentool notwendigerweise verwendet, im Browser derzeit nur sehr schlecht unterstützt werden. So gab es dafür auch

keine direkte Unterstützung im GWT, mit der Folge, dass aufwendige Recherche für eine mögliche Lösung betrieben werden musste.

Bei der Migration habe ich festgestellt, dass man auf der einen Seite sehr viel Code ändern muss und auf der anderen Seite auch sehr viel Code erhalten kann. Das klingt zunächst widersprüchlich, aber gemeint ist folgendes: Erhalten werden konnte nahezu die gesamte Funktionalität des Systems. Darunter verstehe ich die Zusammenhänge der einzelnen Klassen, sowie die logische Funktionalität fast aller Methoden. Trotzdem musste fast alles geändert werden, da das GWT die verwendeten SWT Klassen nicht unterstützt. Diese Änderungen sind verhältnismäßig aber sehr gut zu bewältigen, da stets nur ermittelt werden muss, welche im GWT unterstützte Klasse verwendet werden kann, um eine nicht unterstützte Klasse zu ersetzen und danach einige kleine Unterschiede der Klassen auszugleichen. Problematisch wird es erst dann, wenn es – wie beim JSGraphics besprochen – keine solche äquivalente Klasse im GWT gibt.

Etwas mehr Aufwand verursacht die GUI, denn hier habe ich festgestellt, dass die Anpassung nicht ohne weiteres portierbar ist, da in der Eclipse RCP die bereitgestellten Views und Editoren verwendet werden können und solche Abstraktionsmaße im GWT nicht zur Verfügung stehen.

Insgesamt kann ich sagen, dass ein solches Migrationsvorhaben gut durchführbar ist, dass man sehr viel seines Ursprungscodes in der Funktionalität erhalten kann, dass man aber auch sehr viel Recherchieren muss, um herauszufinden, welche Klassen für andere Klassen verwendet werden können und um Alternativen zu finden.

7) Fazit und Ausblick

Zum Ende meiner Arbeit möchte ich die verwendeten Technologien diskutieren und auf Verbesserungsmöglichkeiten und weitergehende Untersuchungsmöglichkeiten hinweisen.

Festzustellen ist zunächst, dass die Migration unter Verwendung des GWT gut funktioniert hat, da der Aufbau und die Logik des vorhandenen Codes nahezu komplett erhalten werden konnte.
Ein grundsätzliches Problem der derzeitig verfügbaren Browser in Bezug auf Grafikanwendungen ist die Fähigkeit, Grafikoperationen durchzuführen, denn wie bereits früher in dieser Arbeit besprochen, gibt es keine direkten Möglichkeiten im Browser Linien, Rechtecke, Kreise usw. zu zeichnen. Daher bin ich den Weg über die von Walter Zorn entwickelte JavaScript Bibliothek gegangen. Die Interpolation der Grafiken durch Erstellung einzelner Objekt für zusammengefasste Pixel merkt man der Performanz jedoch deutlich an, da die Rechenzeit schon bei einer noch überschaubaren Anzahl von Zeichenfiguren leidet. Das ist jedoch recht spezifisch für unser Zeichenwerkzeug, denn bei anderen möglichen Anwendungen müssen keine grafischen Objekte diesen Ausmaßes erstellt werden. Eine mögliche Verbesserung stellen die im Kapitel 5.2 diskutierten Vektorgrafiken dar. Voraussetzung ist dabei allerdings, dass Vektorgrafiken von nahezu allen Browsern ohne Plugins unterstützt werden und dass sie dann auch von den Werkzeugen wie dem GWT erstellbar sind, was momentan nicht der Fall ist. Das würde die Performanz vermutlich deutlich verbessern, da der Browser dann konkrete Befehle für das Zeichnen von Objekten hat und keine aufwendige Interpolation stattfinden muss.
Genauer betrachtet hat das GWT seine Arbeit nicht wie erwartet erfüllt, weil die Anwendung beispielsweise im Internet Explorer von Microsoft nicht ordnungsgemäß funktioniert und somit das Ziel der browserübergreifenden Verfügbarkeit fehlgeschlagen ist. Fairerweise muss hier aber angemerkt werden, dass zusätzlich zum GWT die Widget Library und die JavaScript Bibliothek von Walter Zorn eingebettet wurden, so dass der Fehler auch in der Komposition dieser Elemente liegen kann.
Ein weiterer kleiner Nachteil ist, dass ich bei der Verwendung des GWT fast den gesamten Quellcode ändern musste. Das waren zwar meist nur kleine Änderungen wie die Verwendung

einer anderen zum GWT kompatiblen Klasse, aber wünschenswert wäre es, wenn das GWT auch die SWT Klassen unterstützen würde und in entsprechende Browserelemente übersetzen könnte. Aus diesem Grunde lohnt es sich auch, sich weitere Frameworks anzusehen, die dieses möglicherweise bereits machen. Hier sind beispielsweise die Frameworks Java2Script Pacemaker (j2s) sowie die Eclipse Rich Ajax Plattform (RAP) zu nennen. J2s gibt auf seiner Seite an, dass es eine JavaScript Implementation von SWT sowie andere Sprachpakete wie java.lang.* und java.util.* (wie auch das GWT) anbietet und geeignet ist, eine SWT basierte Rich Client Anwendung in eine Rich Internet Application zu überführen (Vgl. [J2S07]). RAP ist ein weiteres Framework, das von Eclipse selber entwickelt wurde, um Ajax Webanwendungen mit Hilfe des Eclipse Entwicklungsmodells, Plugins, den bereits besprochenen Erweiterungspunkten sowie einem Widget Toolkit auf SWT Basis zu entwickeln. Das Projekt befindet sich derzeit noch in der Validierungsphase (Vgl. [RAP07]). Eine direktere Einbindung von SWT benötigt vermutlich noch weniger Arbeit bei der Migration und es wäre eine interessante Aufgabe für eine weitere Arbeit, die verschiedenen Frameworks bezüglich Arbeitsaufwand, Performanz und Anzahl erzeugter Zeilen Quellcode zu vergleichen.
Weiterhin führte die fehlende Kompatibilität des GWT mit der Java Sprachversion 5.0 zu Mehraufwand, weil alle neueren Sprachkonstrukte auf den 1.4 Standard geändert werden musste. Es ist aber anzunehmen, dass in zukünftigen Versionen des GWT auch die neueren Sprachversionen unterstützt werden, so dass eine Migration zusätzlich erleichtert wird.

In Bezug auf das LAssi Projekt ist zu sagen, dass die von mir implementierte Version nur kleine Teile von Ajax nutzt, da das Zeichenprogramm hauptsächlich lokal ausgeführt wird und es keine Interaktion mit dem Server gibt. Dadurch wird insbesondere das XMLHttpRequest Objekt nicht benutzt.
Wenn man größere Teile von LAssi als Webanwendung portiert, kann man sich dafür aber sinnvolle Verwendungen überlegen. So kann jeder Nutzer ein eigenes Profil bekommen und die Daten des Nutzers werden auf dem Server verwaltet. Wenn der Nutzer aufwendigere Daten anfordert, können diese asynchron in Teilen geladen werden, so dass sich ein Desktop Stück für Stück aufbaut. Eine weitere Möglichkeit ist, bei einer Lexikonabfrage bereits Vorschläge von Wörtern zu unterbreiten, wenn der Nutzer gerade erst den Anfang des Wortes eingegeben hat. Im weiteren Sinne erscheint es dann auch durchaus möglich, Daten mit

anderen Nutzern auszutauschen, da die Daten alle in gleichen Tabellen organisiert wären und ein berechtigter Nutzer so auch Daten anderer Nutzer auf seinem Desktop anzeigen könnte. Mit solchen Funktionen würden die Möglichkeiten von Ajax noch wesentlich mehr genutzt.
Die komplette Migration schätze ich als realistisch ein, da der Code von LAssi gut strukturiert ist und da LAssi auch grafisch nicht so aufwendig ist, denn in dieser Hinsicht war das Zeichenprogramm am problematischsten für die Browser. Es werden in den anderen Teilen von LAssi zwar auch zahlreiche Zeichenfunktionen benutzt, zum Beispiel um Karteikarten zu repräsentieren, aber dieses ist ebenfalls mit der JavaScript Bibliothek von Walter Zorn möglich. Etwas aufwendiger wird es vermutlich, die GUI zu übertragen, da LAssi an dieser Stelle auf das hilfreiche Toolkit der Eclipse RCP aufbaut und hier bei der Migration ein etwas höherer Aufwand entsteht.
Weiter zu überprüfen bleibt dann nur noch, warum die vom GWT erstellte Version nicht in allen Browsern ordnungsgemäß funktioniert, da das für eine akzeptable Lösung sicherlich eine Bedingung darstellt.

Quellen und Literatur

[AJA07] Ajax 13 Inc.: *Ajax Launch.* http://ajaxlaunch.com, Stand 16.06.2007.

[BER05] Olaf Bergmann, Carsten Bormann: *AJAX - Frische Ansätze für das Web-Design.* 1. Aufl. Teia Lehrbuch Verlag, 2005.

[BUR06] Ed Burnette: *GWT Widget List, 2006.* http://gwtpowered.org/#Widgets, Stand 12.07.2007.

[DAU07] Berthold Daum: *Rich-Client-Entwicklung mit Eclipse 3.2. Anwendungen entwickeln mit der Rich Client Platform.* 2., aktualisierte Auflage dpunkt.verlag, 2007.

[ECL07] The Eclipse Foundation: *About the Eclipse Foundation.* http://www.eclipse.org/org/, Stand 18.07.2007

[GAR05] Jesse James Garrett: *Ajax: A New Approach to Web Applications,* Februar 2005. http://www.adaptivepath.com/publications/essays/archives/000385.php, Stand 17.06.2007

[GJR07] Google: *JRE Emulation Library.* http://code.google.com/webtoolkit/documentation/jre.html, Stand 20.07.2007

[GOO07] Google: *Google Maps.* http://maps.google.de, Stand 16.06.2007.

[GWT07] Google: *Google Web Toolkit, Version 1.3.* *http://code.google.com/webtoolkit/download.html, Stand 15.06.2007.*

[HAN07] Robert Hanson: *Coding SVG With GWT.* http://roberthanson.blogspot.com/2006/06/coding-svg-with-gwt.html, Stand 17.07.2007

[J2S07] Java2Script Pacemaker: *Java2Script: Bridge of RCP to RIA.* http://j2s.sourceforge.net/index.html, Stand 19.07.2007

[LAP06] LAssi-Projekt:*Was sind die nächsten Entwicklungsschritte?* http://www.lassitools.org/cms/index37ab.html?id=59, Stand 21.07.2007

[LAS06] LAssi-Projekt:*Was sind die nächsten Entwicklungsschritte?* http://www.lassitools.org/cms/index4944.html?id=48, Stand 18.07.2007

[MIN07] Stefan Mintert, Christoph Leisegang: *Ajax Grundlagen, Frameworks und Praxislösungen.* 1. Auflage dpunkt.verlag, 2007.

[RAP07] The Eclipse Foundation: *Rich Ajax Plattform (RAP) Project.* http://www.eclipse.org/rap/, Stand 19.07.2007

[STE07] Ralph Steyer: *Das Google Web Toolkit.* Frankfurt: entwickler.press, 2007.

[TAU06] Lutz Tautenhahn: *SVG-VML-3D 1.3.* http://www.lutanho.net/svgvml3d/index.html, Stand 02.07.2007

[WID07] Verschiedene Autoren: *GWT Widget Library,* Version 0.1.4. http://sourceforge.net/project/showfiles.php?group_id=169692, Stand 12.07.2007.

[WIK07] Unbekannt: *Ajax – aus der freien Enzyklopädie Wikipedia.* http://de.wikipedia.org/Wiki/Ajax, Stand 17.06.2007

[ZOR06] Walter Zorn: *DHTML, JavaScript*
Linie, Ellipse, Kreis, Rechteck, Polygon zeichnen.
http://www.walterzorn.de/jsgraphics/jsgraphics.htm, Stand 02.07.2007