Bachelor Thesis

Entwicklung einer Multi-Touch-Anwendung zur interaktiven Nutzung von geografischen Daten

9. Juni 2010

eingereicht von:	Max Ludwig
Matrikelnummer:	521788
Studiengang:	Angewandte Informatik (Bachelor)
	Hochschule für Technik und Wirtschaft Berlin
	Fachbereich 4 - Wirtschaftswissenschaften II
Betreuer:	Prof. Dr. Jürgen Sieck
	Dr. Joachim Quantz

Danksagung

Ich danke:
[Namen folgen]

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Land- und Stadtkarten sind für die Vermittlung von Infomationen, beispielsweise in Museen, auf Messen und im Tourismus, oder bei der Koordination verschiedenster Szenarien, zum Beispiel großangelegten Veranstaltungen oder Einsätzen zur Forschung oder humanitären Hilfe, quasi unerlässlich. Jedoch ist deren Handhabung oft umständlich. Schnell muss zwischen verschiedenen Maßstäben gewechselt werden oder Positionsangaben entnommen und weitergegeben werden. All dies gestaltet sich unter Verwendung herkömmlicher Karten zeitaufwändig und kompliziert.

Eine Multi-Touch-Technologie, welche eine schnelle und natürliche Steuerung ermöglicht, und das Projekt OpenStreetMap, welches seine geografischen Daten für jeden frei zugänglich anbietet, stellen ideale Voraussetzungen für die Realisierung einer solchen Anwendung dar.

1.2 Zielsetzung

Ziel dieser Arbeit ist es, ein System zu entwickeln, welches unter Verwendung einer Multi-Touch-Technologie und den frei zugänglichen Daten des OpenStreetMap-Projektes die Nutzung von geografischen Karten vereinfacht. Im Vordergrund steht dabei die Planung und Realisierung einer interaktiv nutzbaren Karte.

1.3 Aufbau der Arbeit

Zu Beginn werden grundlegende Aspekte, unter anderem die wichtigsten Multi-Touch-Technologien, erläutert. Anschließend wird diskutiert, welche Anforderungen an das zu entwickelnde System gestellt werden. Basierend darauf und abhängig von der Anwendungsumgebung wird eine Softwarearchitektur erstellt. Das Kapitel Implentierung beschreibt die Entwicklung eines Prototypen dieses Systems.

Darauffolgend wird die Funktionalität des Prototypen demonstriert und ausgewertet. Abschließend wird eine Zusammenfassung gegeben. Außerdem werden mögliche Erweiterungen für das System in Aussicht gestellt.

2 Grundlagen

In diesem Kapitel sollen grobe Grundlagen über die Hauptbestandteile der Arbeit beschrieben werden. Es gibt also zum einen eine Einführung in den Bereich „Multi-Touch" sowie zu den wichtigsten Touch-Technologien und zum anderen wird der Begriff „Geoinfomationssysteme" in die Zusammenhänge der Arbeit eingeordnet.

2.1 Multi-Touch

Neben den herkömmlichen Eingabemöglichkeiten wie Tastatur und Maus (oder auch dem Mikrofon bei der Sprachsteuerung) setzt sich seit ein paar Jahren schon die Eingabe/Steuerung über Touchscreens durch, wenn auch momentan zum Großteil nur auf mobilen Endgeräten (Android-Geräte, iPhone, iPod touch und zahlreiche weitere). Am weitesten verbreitet sind auf diesen Geräten kapazitive und resistive Touchscreens. Weiterhin gibt es optische - diese sind auf größerflächigen Touch-Installationen verbreitet - und akustische Verfahren.

Die Interaktion geschieht dabei über bestimmte Gesten, die auch in diesem Kapitel angeschnitten werden.

2.1.1 Touch-Technologien

Im Folgenden werden die wichtigsten und in vielen Aspekten unterschiedlichen Touch-Technologien vorgestellt und verglichen.

kapazitives Verfahren

Beim kapazitiven Verfahren wird die Touch-Oberfläche mit einem Gitternetz aus leitendem Material durchzogen, das ein elektromagnetisches Feld erzeugt. Kommt nun ein weiteres leitendes Material (zum Beispiel ein Finger) in die Nähe des Feldes, verändert es sich und somit auch die Ströme, die durch die Streifen des Gitternetzes fließen. Die Änderung der Ströme wird an den Enden der Streifen des

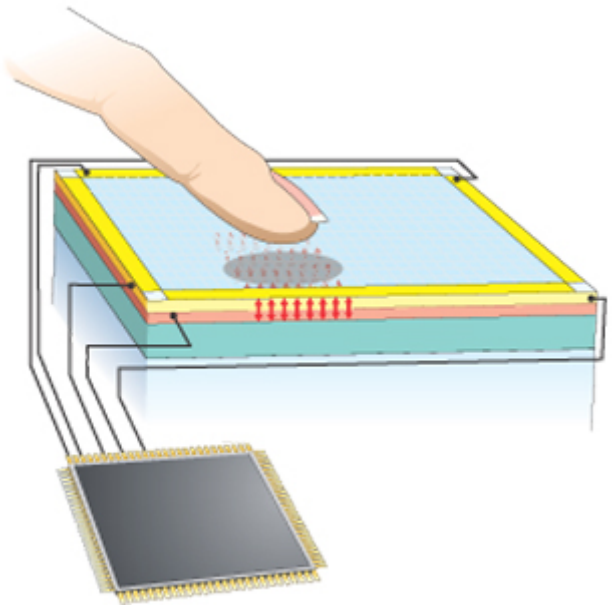

Abbildung 2.1: Schema zur Funktionsweise von kapazitiven Touchscreens [12]

Gitternetzes gemessen. Die x- und y-Position einer Änderung des Feldes, also die Position der Berührung, kann somit festgestellt werden.

Kapazitive Screens sind reaktionsschneller, fast voll lichtdurchlässig und können mit jedem nicht-leitenden Material einfach gereinigt werden, ohne dass unbeabsichtigt eine Eingabe passiert. Nachteile sind eine geringere Genauigkeit und höhere Herstellungskosten gegenüber resitiven Screens.

resistives Verfahren

Resistive Touchscreens werden über physischen Druck bedient. Dazu werden zwei Schichten leitfähiger Folie übereinander gelegt. Winzige Trennpunkte sorgen für einen nötigen Abstand. An zwei gegenübenliegenden Rändern einer der beiden Folien wird nun eine Spannung angelegt, die von einem Rand zum anderen hin kontinuierlich abnimmt. Drückt man nun auf die Oberfläche, wird die Spannung von der einen Folie zur anderen durchgeleitet. Da die Spannung von einer Seite zur anderen abnimmt, kann man anhand der Höhe der Spannung die Position auf der einen Achse feststellen. Nur Millisekunden später wird eine ebenfalls kontinuierlich abnehmende Spannung an die anderen beiden Ränder der Folie angelegt, sodass auch die Position auf der anderen Achse bestimmt werden kann.

Resistive Touchscreens haben den Vorteil, mit dem Finger oder mit einem Stift

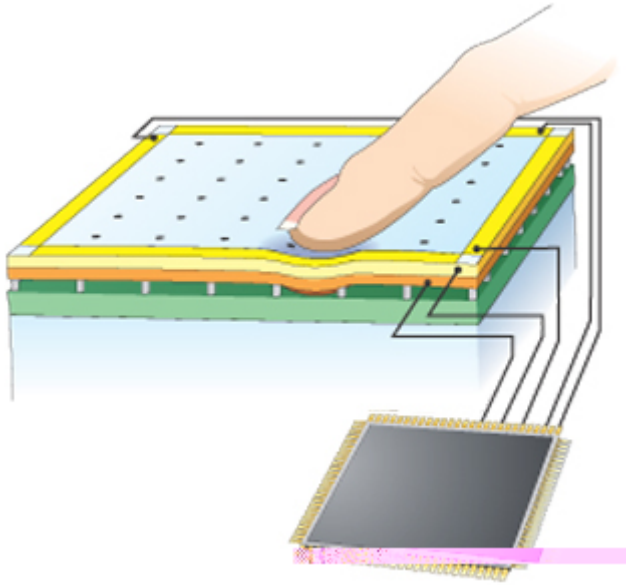

Abbildung 2.2: Schema zur Funktionsweise von resistiven Touchscreens [12]

bedienbar zu sein. Außerdem sind sie relativ preiswert in der Herstellung und bieten Multi-Touch-Eingabemöglichkeiten. Auf der anderen Seite ist die Oberfläche nicht voll lichtdurchlässig und empfindlich gegenüber groben äußeren Einwirkungen.

optisches Verfahren am Beispiel von FTIR (frustrated total internal reflection)

Bei der verhinderten Totalreflexion (engl. *frustrated total internal reflection*, kurz *FTIR*) wird ein optisches Phänomen ausgenutzt, um Berührungen mit der Touch-Oberfläche sichtbar zu machen.

Der Screen besteht unter anderem aus zwei Schichten eines lichtdurchlässigen Materials (z.B. Glas oder Acryl) mit einer bestimmten Brechzahl. Zwischen diese wird Licht gestreut. Das Licht trifft nun in einem bestimmten Winkel, der eine kritische Gradzahl überschreitet, auf eine der Schichten und wird total reflektiert, wenn sich dahinter ein Material mit geringerer Brechzahl (z.B. Luft) befindet. Es ist also zwischen den beiden zuvor beschriebenen Schichten „gefangen". Nähert sich aber ein Material mit höherer Brechzahl als die der beiden Schichten (z.B. ein Finger), wird diese totale Reflexion verhindert, so dass das Licht aus dem Raum zwischen den beiden Schichten „entkommen" kann. Dies wird von einer Kamera, die von unten auf den Screen gerichtet ist, aufgenommen. Anschließend werden die

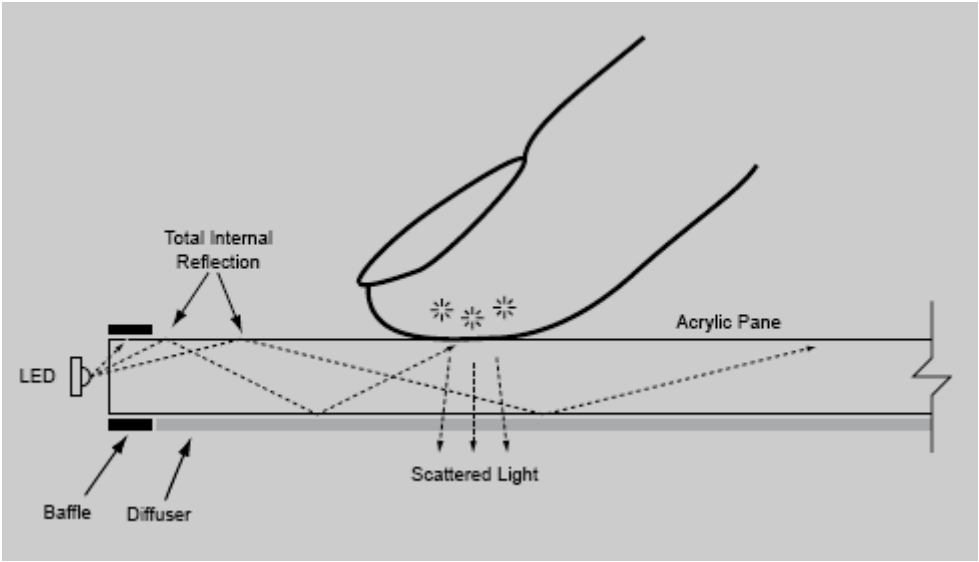

Abbildung 2.3: Schema zur Funktionsweise von Touchscreens, die FTIR verwenden [4]

von der Kamera aufgenommenen Bilder auf besonders helle Punkte - also Stellen, wo eine Berührung statt fand - hin analysiert.

Optische Touchscreens bieten eine hohe Skalierbarkeit und sind sehr dauerhaft, da sie nur mit Glas und Licht arbeiten und somit kaum verschleißanfällig sind. Ein Nachteil ist aber, dass sie durch sich verändernde Lichtverhältnisse, zum Beispiel unter freiem Himmel, beeinflussbar sind.

akustisches Verfahren am Beispiel von SAW (surface acoustic waves)

In diesen Touchscreens werden akustische Wellen durch eine klare Glasoberfläche gesendet. Bei einer Berührung werden diese Wellen absorbiert. Um eine Änderung der Wellen festzustellen, werden sogenannte Interdigitaltransducer benutzt. Diese dienen dazu, akustische Wellen in elektrische Signale umzuwandeln. Eine Änderung der Wellenstärke bedeutet daher auch eine Änderung der elektrischen Signale, die dann ausgewertet werden können.

Wie auch bei optischen Screens bieten diese Screens eine äußerst hohe Lichtdurchlässigkeit. Außerdem sind sie relativ unempfindlich gegenüber äußeren Einwirkungen. Schmutz auf der Oberfläche kann hingegen zu unzuverlässiger Funktionalität führen.

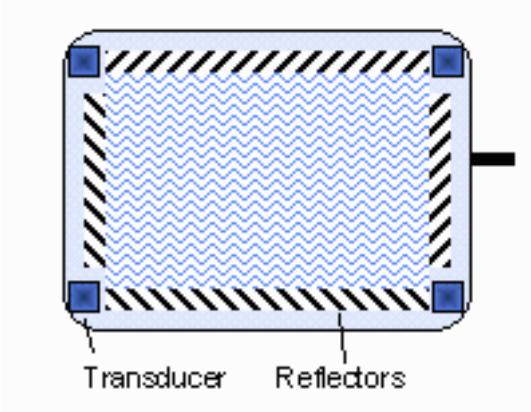

Abbildung 2.4: Schema zur Funktionsweise von Touchscreens, die SAW verwenden [7]

2.1.2 Touch-Gesten

Um mit Touch-Anwendungen zu interagieren, bedarf es einiger vorher fest definierter zeichenhafter Bewegungsabläufe, fortan Gesten genannt, die mit einem Gegenstand, der Hand oder dem Finger auf der Touch-Oberfläche ausgeführt werden müssen.

Unternehmen wie Apple, Microsoft, Palm und Wacom definieren in den Development Guidelines ihrer Produkte die Verwendung von bestimmten Gesten. Apple begründet das in seinen iPhone Human Interface Guidelines ([3]) wie folgt:

> „iPhone users understand these gestures because the built-in applications use them consistently. To benefit from users' familiarity, therefore, and to avoid confusing them, you should use these gestures appropriately in your application."

Es gibt auch schon erste Bemühungen, um bestimmte Gesten mit einer Bedeutung zu vereinheitlichen ([5]).

Die wohl denkbar einfachste Geste ist dabei das einfache Tippen auf die Oberfläche, vergleichbar mit einem Mausklick. Weitere Single-Touch-Gesten wären zum Beispiel eine Berührung für einen längeren Zeitraum, das Wischen von einem Punkt zu einem nächsten und allgemein das Nachzeichnen vielerlei Formen.

Obgleich Single-Touch schon eine beachtliche Anzahl Gesten, auch durch Kombination einzelner, bereitstellt, bietet Multi-Touch noch weit mehr Möglichkeiten. Möglich ist die gleichzeitige Kombination unterschiedlichster Single-Touch-Gesten. Als Beispiel seien hier die bekannten Gesten zum Vergrößern oder Verkleinern von

Objekten, die Hände/Finger bewegen sich voneinander weg oder aufeinander zu, oder das Nachzeichnen eines Kreises mit versetzten Händen/Fingern zum Drehen von Objekten.

2.2 digitale Kartendienste

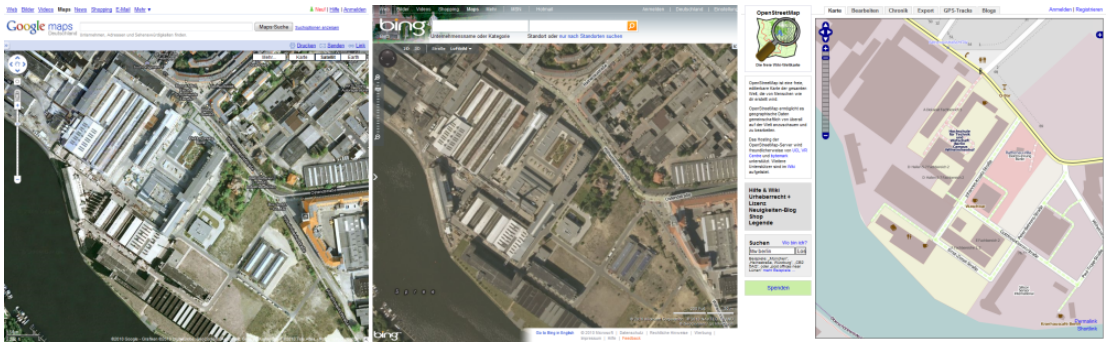

Abbildung 2.5: Google Maps, Bing Maps und OpenStreetMap im Browser [13, 14, 10]

Zu den bekannteren digitalen Kartendiensten gehören Google Maps[1], Bing Maps[2] von Microsoft und *OpenStreetMap* (OSM)[3]. Alle drei ähneln sich stark in Darstellung der Karten und Bedienung. Allerdings ist nur OpenStreetMap ein Geoinformationssystem im eigentlichen Sinne, da zu einem solchen laut Norbert de Lange ([1]) die „Aufnahme, Speicherung, Aktualisierung, Verarbeitung und Auswertung von Informationen sowie deren Wiedergabe" gehören. Google und Bing Maps bieten derlei Möglichkeiten bezüglich Geoinformationen gar nicht oder nur in sehr geringem Maße an. Bei Google Maps können sich beispielsweise Unternehmen, Betriebe und Selbständige mit ihrer Firmenadresse eintragen.

OpenStreetMap hingegen gewährt freien Zugang zu seinen gesamten geographischen Daten. Man kann neue Daten wie zum Beispiel Straßen, Wege, Plätze, Gebäude und besondere Orte wie Sehenswürdigkeiten hinzufügen, sie ändern und direkt im System speichern, so dass sie für den Rest der Nutzerschaft ebenso zugänglich sind.

[1] http://maps.google.de/
[2] http://www.bing.com/maps/
[3] http://www.openstreetmap.org/

3 Anforderungsanalyse

Dieses Kapitel soll Anforderungen an das System (im folgenden *Miami, **M**iami **i**s **a** **m**ap **i**nterface,* genannt) aufzeigen und detaillierte Anwendungsfälle nennen.

3.1 Anforderungen

3.1.1 Kartendarstellung

Die Karte soll möglichst hochauflösend und bei Interaktion flüssig dargestellt werden.

Sieht man sich Kartendienste für Browser oder Smartphones an, fällt sofort auf, dass die Karte oft nicht flüssig dargestellt wird, da Kartenkacheln nicht schnell genug nachgeladen werden. Grund dafür ist die meist geringe Bandbreite der Internetverbindung. Da Smartphones nur geringe Speicherkapazitäten haben, kommt eine lokale Speicherung der Kartenkacheln nicht in Frage. Der Touchmaster verfügt hingegen über genügend und auch erweiterbare Kapazitäten, wodurch eine lokale Datenhaltung Vorteile hat. Bei Smartphones würde es sich außerdem negativ auf die Systemperformance und die Kosten, die durch den Internetverkehr entstehen, auswirken, wenn man schon vor einer Interaktion mehr Kartenkacheln vorlädt als für die sichtbare Fläche nötig. Bei einer lokalen Datenhaltung entfallen diese Nachteile und durch das Vorladen von Kacheln, die sich nicht im sichtbaren Bereich befinden, wird sichergestellt, dass man ohne sichtbare Unterbrechungen mit der Karte interagieren kann.

Über eine flüssige Kartendarstellung hinaus soll es möglich sein, eine Legende ein- und auszublenden. Außerdem sollen verschiedene Ebenen über die Karte gelegt werden können, um ihren Informationsgehalt anzureichern. Denkbar wären hier zum Beispiel eine farbcodierte Darstellung von Höhenunterschieden oder eine Hervorhebung von sogenannten „points of interest” (POI) durch Beschriftungen.

3.1.2 Karteninteraktion

Die Interaktion mit der Karte ist ein wesentlicher Bestandteil der Anwendung, wenn nicht sogar der wesentlichste. Es müssen verschiedene Gesten zu unterschiedlichen Interaktionen festgelegt werden. Grundlegende Interaktionen wie das Verschieben der Karte sowie das Vergrößern und Verkleinern des Kartenausschnitts (Zoomen) sollten auf jeden Fall implementiert werden. Neben diesen Interaktionen gibt es weitere, die denkbar und auch alle in Anwendungen auf Smartphones zu finden sind, wie das Rotieren der Karte oder das Markieren oder Auswählen von bestimmten Punkten, zum Beispiel POIs. Das Anzeigen und Wechseln bestimmter Ebenen mit Zusatzinformationen für die Karte ließe sich auch mit bestimmten Gesten realisieren.

Um den Einsatz des Systems nicht zu sehr zu beschränken, soll die gesamte Steuerung außerdem über eine Maus möglich sein.

3.2 Use-Cases

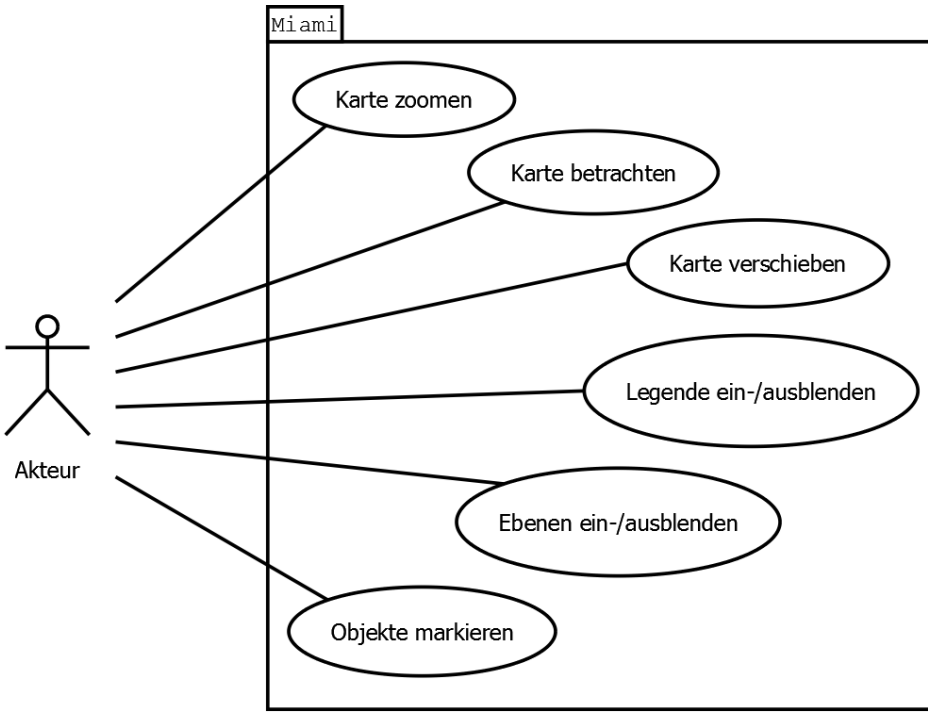

Abbildung 3.1: Use-Case-Diagramm, das die möglichen Anwendungsfälle übersichtlich darstellt

Wie im vorangegangenen Abschnitt erörtert, stellt nun Abbildung 3.1 die wichtigesten Anwendungsfälle in einem Diagramm übersichtlich dar.

Neben den offensichtlichen Anwendungsfällen wie dem Betrachten, Verschieben oder Zoomen der Karte, ergeben sich noch weitere Möglichkeiten, mit dem System zu interagieren. Durch Gesten oder Bedienelemente könnten eine Legende oder bestimmte Ebenen zur Hervorhebung von thematisch zusammenhängenden Inhalten ein- und ausgeblendet werden. Weiterhin wäre es möglich, der Karte Markierungen hinzuzufügen.

4 Systementwurf

Im Folgenden werden die verwendeten Komponenten des Systems beschrieben.

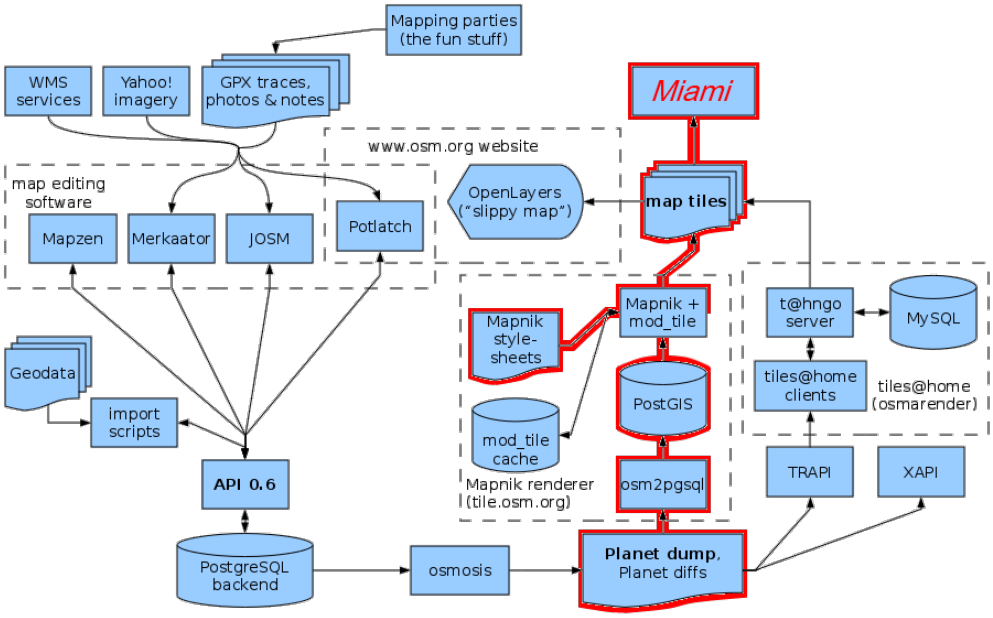

Abbildung 4.1: OpenStreetMap-Komponentendiagramm (Komponenten, die in Zusammenhang mit dieser Arbeit genutzt wurden und werden, sind hervorgehoben) [10, bearbeitet]

Abbildung 4.1 zeigt nahezu alle Komponenten der OpenStreetMap-Infrastruktur sowie Systeme und Tools verschiedener Projekte, die mit den geographischen Daten arbeiten. Karteneditoren wie der in Java geschriebene JOSM oder auch der über den Browser zugängliche Flash-basierende Potlach greifen im Gegensatz zu Miami direkt über die OSM-API[1] auf die in einer PostgreSQL-Datenbank liegenden Geodaten zu und bekommen diese im XML-Format. Die API basiert auf den Ideen der RESTful-API, wird aber hier nicht näher erläutert, da es den Rahmen der Arbeit sprengen würde.

[1] http://wiki.openstreetmap.org/wiki/Api

Die Daten für Miami entstammen aus Dumps der OSM-Datenbank. OpenStreetMap generiert wöchentlich Dumps des gesamten Planeten („Planet dump") im XML-Format, allerdings liegen auch Dumps für einzelne Länder vor. Ausgehend von diesen Dumps gibt es 2 etablierte Wege, um Kartenkacheln („map tiles") zu generieren. Diese müssen zuvor von einem Renderer erstellt werden. Dazu kann entweder *Mapnik*[2] oder *Osmarender* verwendet werden. Ersteres ist ein „Free Toolkit for developing mapping applications" und wird für die *Slippy Map* (die Weboberfläche von OSM) eingesetzt, Letzterer ist ein „rendering tool for generating SVG images of OSM data" und wird von *tiles@home*, einem System, bei dem die Grafiken verteilt auf vielen Heimrechnern gerendert werden, genutzt.

4.1 Anwendungsumgebung

In diesem Abschnitt wird dargelegt, in welcher Umgebung das System laufen und entwickelt wird. Das heißt, welche Frameworks und Programmiersprachen benutzt wurden, auf welcher Plattform entwickelt wurde und auf welcher das System laufen wird.

4.1.1 Software

Betriebssystem und Programmiersprache

Das genutzte Betriebssystem ist die Linux-Distribution Ubuntu in der Version 9.04. Zwar sind alle Komponenten, wie Mapnik oder *Y60*, laut Entwicklern auch unter Windows oder Mac lauffähig, jedoch bietet Ubuntu sämtliche Abhängigkeiten, wie zum Beispiel Bibliotheken für Mapnik und Y60, vorkompiliert über die Paketquellen an. Zudem ist Ubuntu meine präferierte Arbeitsumgebung.

Als Programmiersprache kommt JavaScript zum Einsatz, da *Spark*, das genutzte Framework, darauf basiert.

Frameworks

Y60 wird vom Entwickler ART+COM selbst als „3D-Multimedia-Anwendungsplattform" bezeichnet. Es ist in C++ geschrieben und bietet eine Anbindung für JavaScript.

Spark ist ein auf JavaScript-basierendes Komponenten-Framework, welches auf Y60 aufsetzt.

[2] *http://mapnik.org/*

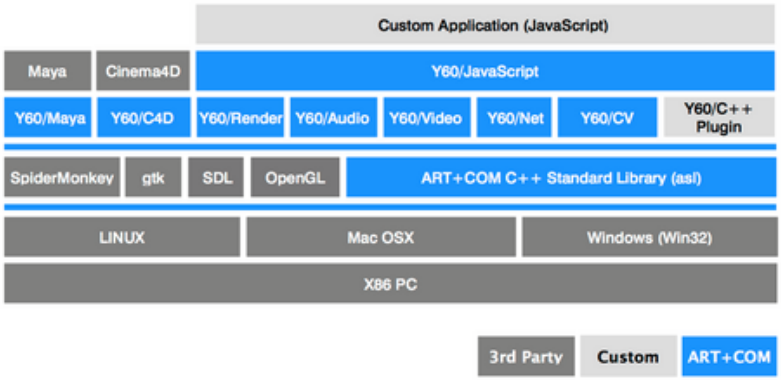

Abbildung 4.2: Schema, das Y60 in die Anwendungsumgebung einordnet [15]

Da ich im Rahmen meines Praktikums Erfahrung mit Spark und Y60 gesammelt habe, verwende ich beides nun auch in dieser Arbeit.

Abbildung 4.2 zeigt, wie sich Y60 in eine Umgebung einordnet. Auf einem Rechner mit x86-Architektur und Linux, Windows oder Mac OS X als Betriebssystem kann Y60 unter Voraussetzung bestimmter Bibliotheken kompiliert werden. Auf Y60 können dann Anwendungen oder Frameworks wie Spark aufsetzen.

Tools

Zum Rendern der Kartenkacheln kann man entweder selbst einen Renderer schreiben, was sich aber als sehr aufwändig und umfangreich erweisen würde, oder man bedient sich eines externen Tools. Hier bieten sich zwei Alternativen zum Schreiben eines eigenen Renderers an. Zum einen gibt es Mapnik und zum anderen Osmarender.

Ersterer wird von OpenStreetMap verwendet, um die sogenannte Slippy Map (das Karteninterface für den Webbrowser) mit Rastergrafiken zu versorgen. Da Mapnik nicht direkt mit dem OpenStreetMap-Format arbeiten kann, müssen die Daten mit dem Tool *osm2pgsql* konvertiert und in Mapniks Datenmodell, auf welches an dieser Stelle aber nicht näher eingegangen wird, geladen werden. Mapnik lädt dann die Daten, rendert Kartenkacheln fester Größe und speichert sie in einer bestimmten Ordnerstruktur im Dateisystem.

Osmarender hingegen rendert anhand von Stylesheets Vektorgrafiken im SVG-Format. Nachteil dabei ist, dass Spark die Darstellung von SVG-Dateien nicht unterstützt und diese erst konvertiert werden müssten.

Das Schreiben eines eigenen Renderers stellt einen ungerechtfertigt hohen Auf-

wand dar. Da die SVG-Dateien von Osmarender erst konvertiert werden müssen und Mapniks Tools die Kacheln schneller rendern und unmittelbar in die benötigte Verzeichnisstruktur speichern, wird für dieses System Mapnik verwendet.

4.1.2 Hardware

Die Anwendung kann über viele Multi-Touch-Geräte, wie touch-fähigen Bildschirmen usw., gesteuert werden. Voraussetzung ist lediglich, dass die Signale der *Proximatrix 400* entsprechend an Y60 weitergeleitet werden.

Touchmaster und Proximatrix 400

Abbildung 4.3: Touchmaster von ART+COM [15]

Der *Touchmaster* ist ein Multi-Touch-Präsentationssystem der Firma ART+COM. Wie in Abbildung 4.3 zu sehen, verfügt er über eine großformatige Tischoberfläche, auf die per Beamer über zwei Spiegel - der eine direkt über dem Tisch montiert - das Bild projiziert wird. Ein Controller ist mit einem Computer und der in der Tischplatte befindlichen Proximatrix 400 verbunden, um die Signale an den Treiber weiterzuleiten.

Die Proximatrix 400, die beim Touchmaster (also dem Gerät, auf dem der Prototyp laufen wird) zum Einsatz kommt, verwendet kein optisches Erkennungsverfah-

ren. Stattdessen befindet sich in der Platte aus nicht-leitendem Material ein Gitter leitenden Materials mit 4 Kreuzungspunkten pro Dezimeter. An die Enden des Gitters wird eine Spannung angelegt. Der Controller liefert über USB oder Ethernet 20 mal pro Sekunde Messwerte für jeden Kreuzungspunkt. Diese werden von einem Treiber ausgewertet und an die Anwendung weitergeleitet. Sobald ein leitendes Material (z.B. ein Finger oder eine Hand) nur in die Nähe der Platte kommt, verändert sich das elektromagnetische Feld des Gitters und der Treiber liefert entsprechende Daten, aus denen man die Position der Interaktion errechnen kann. Das Ergebnis ist durch die relativ große Entfernung zwischen den Kreuzungspunkten etwas ungenau.

4.2 Architektur

Die Anwendung soll nach Möglichkeit auf dem bewährten Model-View-Controller-Konzept (MVC) beruhen. Auch Spark als Framework legt die Nutzung von MVC nahe.

4.2.1 Modellschicht

Die zu haltenden Daten sind fast ausschließlich die von Mapnik vorgerenderten Kartenkacheln. Obwohl diese auch von einem entfernten Server bereitgestellt werden könnten - OpenStreetMap bietet mit `tile.openstreetmap.org` einen solchen sogenannten Tile-Server an -, befinden sie sich in diesem Fall im lokalen Dateisystem.

Bei der Verwendung von entfernten Servern als Datenhaltung muss auf die Verbindung zu eben diesen geachtet werden. Eine instabile oder bezüglich der Bandbreite schwache Verbindung würde dazu führen, dass Kacheln während der Interaktion mit der Karte nicht schnell genug nachgeladen werden und so keine flüssige Darstellung mehr möglich ist.

Um die Kacheln lokal zu rendern, ist es nötig einen Dump der OpenStreetMap-Datenbank mit dem Tool osm2pgsql in die von Mapnik genutzte Datenbank zu migrieren. Diese Datenbank ist Bestandteil des freien Geoinformationssystems PostGIS[3].

> „PostGIS adds support for geographic objects to the PostgreSQL object-relational database. In effect, PostGIS "spatially enables" the PostgreS-

[3] http://postgis.refractions.net/

> QL server, allowing it to be used as a backend spatial database for geographic information systems (GIS) [...]" [11]

Sie enthält alle nötigen Daten, mit denen Mapnik mithilfe der AGG-Bibliothek[4] 256x256 Pixel große Kartenkacheln im 32bit-PNG-Format rendern kann.

Weiterhin werden Metadaten in einer XML-Datei gehalten. Diese Daten repräsentieren die Felder im Gitter für die Kacheln. Es werden sowohl x- und y-Werte für die relative Position im Gitter als auch Spalte und Reihe, in der sich die Kachel im Gitter befinden wird, in einem XML-Tag festgehalten. Abbildung 4.4 zeigt eine schematische Darstellung der Metadaten. Neben den durchnummerierten Kachelcontainern sieht man einen blau-transparenten Bereich, in dem die Karte sichtbar sein wird und in dem Gesten ausgeführt werden können. Die Namen der Kachelcontainer müssen streng monoton um den Wert 1 steigen, da später ein Array die Verwaltung aller Container übernimmt. Die roten Zahlen links und unten stellen die Reihen und Spalten des Kachelgitters dar. Diese dienen dazu, eine Kartenkachel dem richtigen Container zuzuordnen.

4.2.2 Präsentationsschicht

Die Präsentationsschicht wird von den SPARK-Dateien übernommen. Diese bedienen sich der XML, um Elemente für die Nutzeroberfläche zu deklarieren.

Die Nutzeroberfläche enthält mindestens die Karte, die sich aus den gerenderten Kartenkacheln zusammensetzt. Um diese anzuordnen, wird ein Gitter erstellt, in dessen Felder die Kacheln geladen werden (siehe Abbildung 4.4). Um die Rechnerressourcen nicht übermäßig zu belasten, was geschehen würde, wenn man sämtliche Kacheln in ein Gitter lädt, das so groß ist wie die Karte, die man darstellen will, ist das Gitter nur um eine Spalte und Reihe zu jeder Seite größer als zur vollständigen Darstellung des Kartenausschnitts notwendig ist. Dadurch wird auch sichergestellt, dass beim Verschieben der Karte keine „leeren" Flächen durch eventuelle Latenzen beim Laden der nächsten Kacheln zustande kommen.

Weiterhin lassen sich Ebenen, die Inhalte wie Gebiete, bestimmte Wegstrecken oder thematisch zusammenhängende Orte von Interesse hervorheben sowie eine Legende ein- und ausblenden.

Der Nutzer interagiert entweder über Bedienelemente oder Gesten mit der Anwendung. Die Gesten orientieren sich an die bereits im Grundlagenkapitel erwähn-

[4] http://www.antigrain.com/

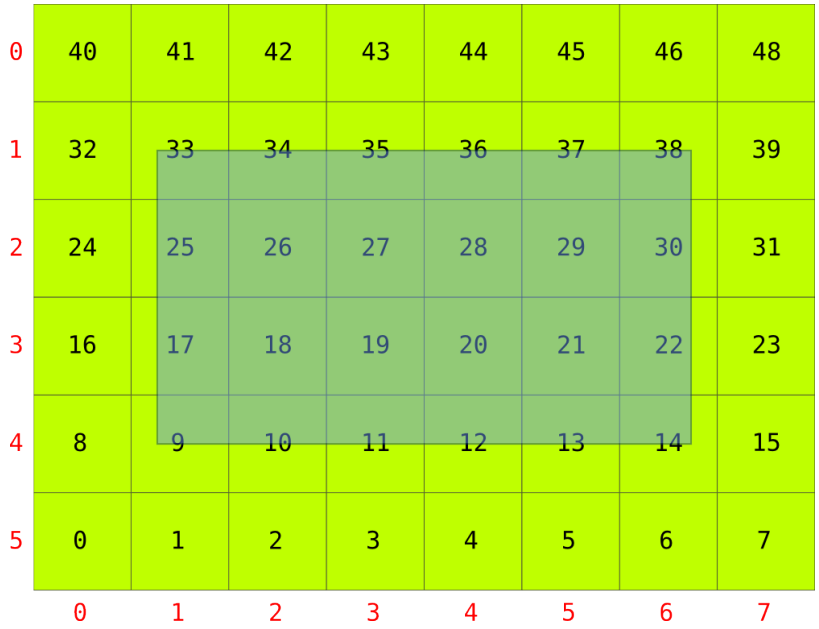

Abbildung 4.4: schematische Darstellung der Metadaten zum Kachelgitter

ten Standardgesten.

Das Verschieben der Karte geschieht durch eine Wischbewegung, bei der die Karte dem Berührungspunkt folgt. Das Zoomen der Karte geschieht durch ein Aufeinanderzu- oder Voneinanderwegbewegen zweier Berührungspunkte.

4.2.3 Steuerungsschicht

So gut wie jede SPARK-Datei hat ihre Implementation. Diese tritt als Controller auf. Sie nimmt durch den Nutzer ausgelöste Interaktionsereignisse entgegen und verarbeitet sie. Dem Ereignis entsprechend werden dann Daten im Modell angepasst oder Eigenschaften der Elemente in der Präsentationsschicht geändert.

Wird die Kartenoberfläche berührt, wird die Kartenposition entsprechend der des Berührungspunktes geändert. Da die Erkennung der Position des Berührungspunktes etwas grob ist, wird ein Näherungsalgorithmus verwendet. Dieser merkt sich eine bestimmte Menge Bewegungsänderungen und errechnet aus allen Werten den Durchschnitt, wodurch eine interpolierte Position zustande kommt. Die Aktualisierung der Kartenposition liegt dadurch zeitlich minimal hinter der des Berührungspunktes.

In Abbildung 4.5 wird dargestellt wie eine Interaktion mit Miami abläuft. Nachdem Der Nutzer den Tisch berührt hat, wird ein Event abgeschickt. Wird dieses als

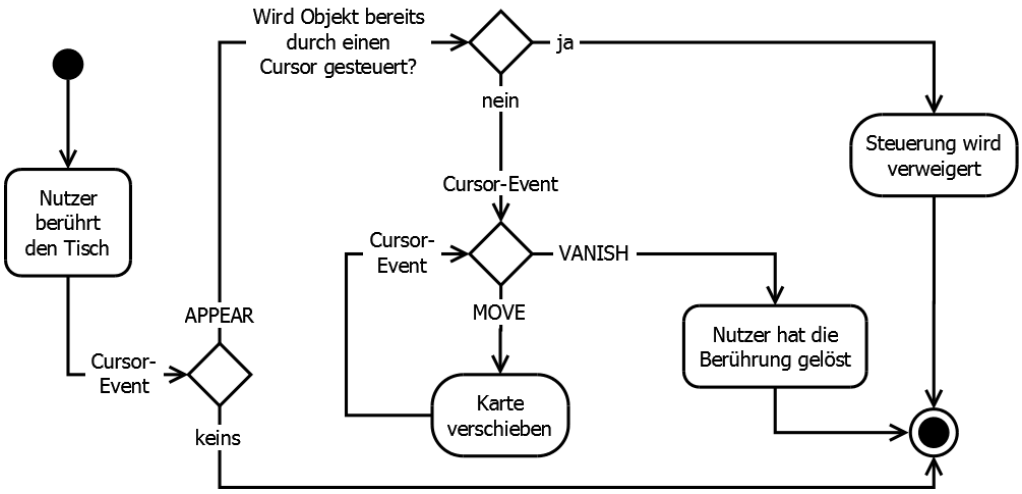

Abbildung 4.5: Diagramm, dass den Ablauf einer Interaktion mit Miami darstellt

`APPEAR`-Event vom EventListener des Kachelgitters abgefangen, wird zuerst geprüft, ob schon ein Cursor existiert, der das Kachelgitter steuert. Ist dies der Fall, wird die Steuerung verweigert. Andernfalls wird die ID des Cursors gespeichert, um sicherzustellen, dass alle Events vom selben Cursor kommen. Nun werden weitere Events dieses Cursors erwartet. Ein `MOVE`-Event enthält unter anderem die Position des Cursors, so dass die Kartenposition simultan zur Cursorposition geändert wird. Löst der Nutzer die Berührung, wird ein `VANISH`-Event abgeschickt, bei dessen Empfang die gespeicherte Cursor-ID wieder gelöscht wird und so die Karte wieder zur Steuerung durch neue Cursor freigegeben ist.

	Event	**Bedeutung**
Touch-Events	`APPEAR`	bei jedem Entstehen eines Cursors; sprich, wenn der Nutzer den Tisch berührt
	`MOVE`	bei Positionsänderung eines Cursors
	`VANISH`	bei Verschwinden eines Cursors; sprich, wenn der Nutzer die Berührung mit dem Tisch löst
Maus-Events	`BUTTON_DOWN`	wenn ein Mausbutton heruntergedrückt wird
	`MOVE`	bei Positionsänderung der Maus
	`BUTTON_UP`	wenn ein Mausbutton losgelassen wird
	`SCROLL`	beim Scrollen mit der Maus; bei Mausrädern, die einrasten, ist ein Schritt des Rades ein Event

Tabelle 4.1: verwendete Spark-Events und ihre Bedeutung

Tabelle 4.1 listet alle zu verwendenden Events auf und klärt ihre Bedeutung. Miami ist sowohl durch Touch- als auch durch Maus-Eingabe steuerbar. Dabei verhalten sich `BUTTON_DOWN` und `BUTTON_UP` äquivalent zu `APPEAR` und `VANISH`, das heißt, das Verschieben der Karte kann man sowohl durch klassisches Drag'n'Drop per Maus als auch durch Wischen auf der Touch-Oberfläche erreichen.

Wenn die Karte soweit verschoben wird, dass das Laden neuer Kacheln erforderlich wird, wird die Gitterposition komplett zurückgesetzt und jedem Gitterfeld eine neue Kachel zugewiesen.

Beim Zoomen der Karte wird ein völlig neue Kachelebene „betreten". Dadurch ist es nötig, neue Kachelkoordinaten zu berechnen. OpenStreetMap stellt dazu Funktionen zur Umrechnung von Längen- und Breitengraden in x- und y-Koordinaten bereit. Wird die Karte gezoomt, wird zunächst festgestellt, über welcher Kachel das Zoom-Event abgeschickt wurde. Die Kachelkoordinaten werden nun in Längen- und Breitengrade umgerechnet und anschließend dann wieder mit der neuen Zoomstufe in Kachelkoordinaten zurück umgerechnet

5 Implementierung

In diesem Kapitel wird beschrieben, welche Anforderungen an Miami umgesetzt und wie sie implementiert wurden. Die Implementierung hält sich an das Model-View-Controller-Konzept.

Auf Y60 aufsetzende Spark-Anwendungen werden durch eine immer gleiche Einstiegsroutine gestartet. Listing 5.1 zeigt die Routine wie sie in Miami verwendet wird.

Listing 5.1: Einstiegsroutine für Miami

```
1  // Spark importieren
2  use("spark/spark.js");
3  // Klassen von Miami importieren
4  use("MiamiImpl.js");
5  use("Model.js");
6  use("TileGridImpl.js");
7  use("Tile.js");
8
9  try {
10     // Hauptlayout laden
11     var miami = spark.loadFile("LAYOUT/Miami.spark");
12     // Proximatrix aktivieren
13     spark.enableProximatrix(miami);
14     // Eventschleife anstoßen
15     miami.go();
16 } catch (e) {
17     reportException(e);
18     exit(1);
19 }
```

Listing 5.1: Einstiegsroutine für Miami

In Zeile 1 wird das Spark-Framework geladen. Die folgenden `use`-Anweisungen importieren die einzelnen Klassen, die in den nächsten Abschnitten näher beschrieben werden.

Im `try`-Block wird nun zunächst das `Hauptlayout` geladen, wodurch Spark die

darin definierten Klassen instanziiert. Diese Klassen befinden sich wiederum in den zuvor importierten Dateien.

Nachdem in Zeile 11 die Nutzung der Proximatrix für Miami aktiviert wurde, wird mit der `go`-Methode die Hauptschleife angestoßen. Deren Aufgabe ist es, auf Events zu hören, die entstehen, sobald der Nutzer die Proximatrix berührt.

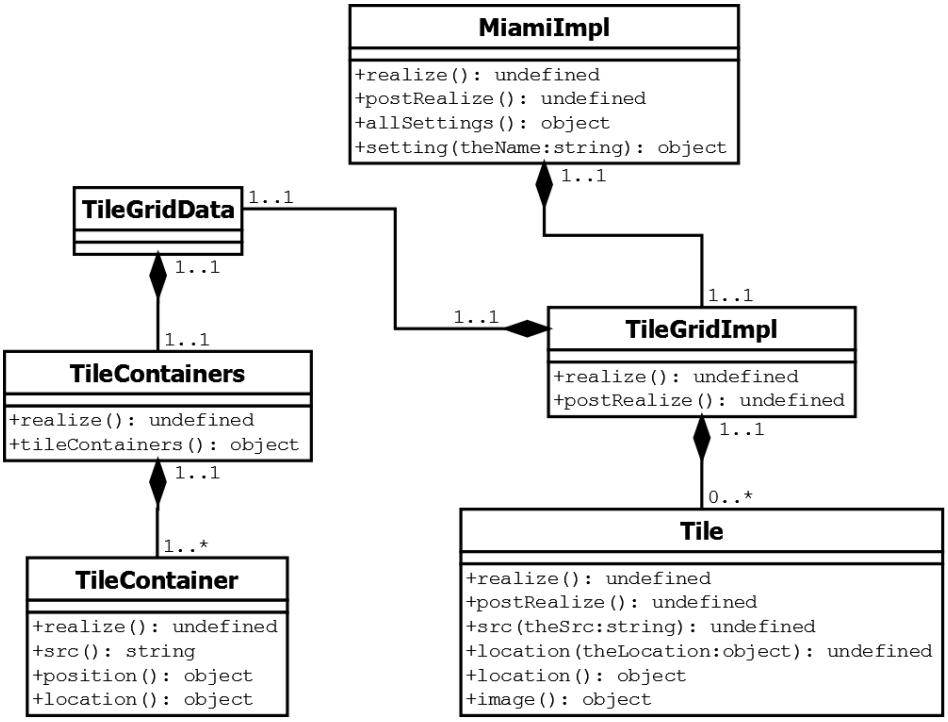

Abbildung 5.1: Diagramm, dass alle Klassen und deren Beziehungen darstellt

Abbildung 5.1 stellt eine Übersicht aller Klassen und ihrer Beziehungen zueinander dar. Während sich die Klassen *MiamiImpl*, *TileGridImpl* und *Tile* in eigenen Dateien befinden, wurden die Klassen *TileGridData*, *TileContainers* und *TileContainer* aufgrund ihrer Zusammengehörigkeit in der Datei *Model.js* zusammengefasst.

5.1 Modellschicht

Wie im Kapitel Systementwurf schon beschrieben, werden die Kartenkacheln von Mapnik gerendert und in eine Verzeichnisstruktur gespeichert, die wie folgt aussieht:

`<Pfad zum Kachelordner>/<Zoomstufe>/<x-Koordinate der Kachel>/<y-Koordinate der Kachel>.png`

Beispiel anhand einer Kachel auf einem OpenStreetMap-Tile-Server (hier der Große Stern in Berlin auf Zoomstufe 16):

http://tile.openstreetmap.org/16/35198/21494.png

Die Metadaten zum Kachelgitter liegen in einem Dokument im XML-Format vor, das wie Listing 5.2 zeigt aufgebaut ist.

Listing 5.2: Metadaten zum Kachelgitter in der Datei *tile_grid.xml*

```
1 <TileGridData>
2   <TileContainers name="tiles">
3     <TileContainer src="IMAGES/notile.png" name="0" col="0" row="5" x="0"
          y="0" z="0" />
4     <TileContainer src="IMAGES/notile.png" name="1" col="1" row="5" x="256"
          y="0" z="0" />
5     <TileContainer src="IMAGES/notile.png" name="2" col="2" row="5" x="512"
          y="0" z="0" />
6     ...
7     <TileContainer src="IMAGES/notile.png" name="47" col="7" row="0"
          x="1792" y="1280" z="0" />
8   </TileContainers>
9 </TileGridData>
```

Listing 5.2: Metadaten zum Kachelgitter in der Datei *tile_grid.xml*

Anhand des `name`-Attributs des umschließenden Tags `TileContainers` werden die einzelnen Felder des Gitters, die `TileContainer`, gefunden. Um die Felder des Gitters initial mit einem Bild zu besetzen wird dem `src`-Attribut der Pfad zu einem Musterbild, das angezeigt wird, wenn eine nicht verfügbare Zoomstufe gewählt oder wenn die Karte über ihre Grenzen hinweg verschoben wurde, zugewiesen.

Alle Knoten dieses XML-Dokuments bilden Objekte der Klassen in der Datei *Model.js* ab, die in der Einstiegsroutine (Listing 5.1) importiert wurde. Die Klassen in *Model.js* stellen die Schnittstelle zum Modell dar. Ihre Instanzen sind die Abbilder der Knoten, die in Listing 5.2 zu sehen sind.

5.2 Präsentationsschicht

Die gesamte Präsentationsschicht wird durch XML-Dokumente repräsentiert. Diese sogenannten SPARK-Dateien beinhalten das Layout.

Listing 5.3: Metadaten zum Kachelgitter

```
1  <MiamiImpl
2    name="miami"
3    title="Miami is a map interface"
4    decorations="false"
5    mouseCursor="false"
6    width="1400"
7    height="770">
8
9    <Template name="TileGrid" src="LAYOUT/TileGrid.spark" />
10
11   <TileGrid name="tilegrid" x="-320" y="-383" width="1792" height="1280"
         z="0" />
12 </MiamiImpl>
```

Listing 5.3: Metadaten zum Kachelgitter

Listing 5.3 stellt das Hauptlayout in der Datei *Miami.spark* dar. Der Tag `MiamiImpl` bildet das Fenster ab, indem sich die Anwendung befindet. Mit dem `Template` Tag wird auf ein externes Layoutdokument namens `TileGrid` verwiesen, welches in Zeile 11 zur späteren Instanziierung deklariert wird.

Die schwarz umrandete klare Fläche in der Mitte von Abbildung 5.2 stellt das Fenster der Anwendung dar. Das Kachelgitter wird darin zentriert. Da es größer als die sichtbare Fläche ist, sind die Positionskoordinaten negativ.

Listing 5.4: Beispiellayout einer Legende zur Karte

```
1 <Transform name="legende" x="0" y="0" z="1" width="150" height="100">
2   <Image name="fluss" src="IMAGES/legende/fluss.png" x="20" y="20" />
3   <NewText name="fluss" text="Fluss" x="50" y="20" />
4   <Image name="strasse" src="IMAGES/legende/strasse.png" x="20" y="50" />
5   <NewText name="strasse" text="Straße" x="50" y="50" />
6 </Transform>
```

Listing 5.4: Beispiellayout einer Legende zur Karte

Listing 5.4 zeigt zur Verdeutlichung beispielhaft ein Spark-Layout, das eine Legende für die Karte darstellt. Dieser Layoutteil müsste ein Kindknoten des `MiamiImpl`-Knotens sein - es ist allerdings nicht implementiert worden.

Der `Transform` Tag stellt einen Container dar, der mit Attributen wie `color` grafisch gestaltet werden kann. Alle Elemente darin werden relativ zu ihm positioniert. Die Tags `Image` und `NewText` dienen zur Anzeige eines Bildes und eines Textes. Dargestellt wird hier also eine Box, die sich in der unteren linken Ecke

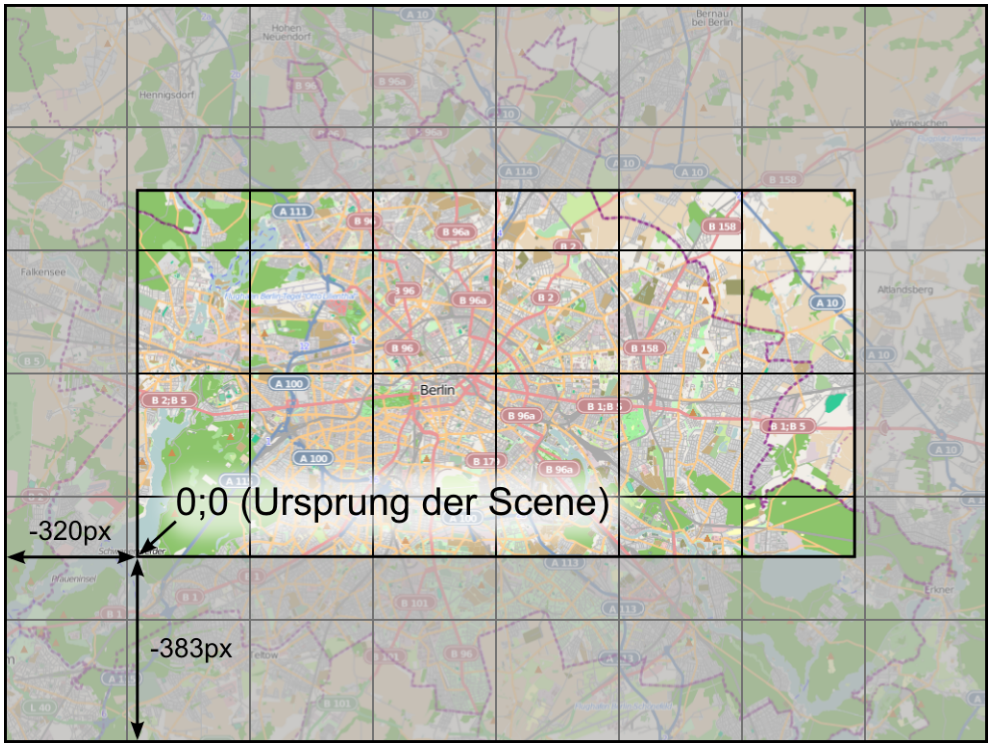

Abbildung 5.2: Schema, das die Positionierung des Kachelgitters zeigt

des Fensters befindet und die auf zwei Zeilen je ein Bild mit einem Text daneben enthält.

5.3 Steuerungsschicht

Die Steuerung übernehmen in JavaScript geschriebene Klassen. Diese werden initialisiert sobald Spark auf ihren korrespondierenden XML-Knoten im Layout stößt. In Listing 5.1 wird Spark mit dem Befehl `spark.loadFile("LAYOUT/Miami.spark")` angewiesen, das Hauptlayout zu laden. Spark analysiert nun den XML-Baum, stößt zuerst auf den `MiamiImpl` Knoten und instanziiert die korrespondierende Klasse (Listing 5.5), die sich in der zuvor importierten Datei *MiamiImpl.js* befindet.

Listing 5.5: MiamiImpl-Klasse, die das Fenster der Anwendung repräsentiert

```
1 spark.MiamiImpl = spark.ComponentClass("MiamiImpl");
2 spark.MiamiImpl.Constructor = function(Protected) {
3     var Base = {};
4     var Public = this;
5     this.Inherit(spark.Window);
```

```
6
7       var _mySettings = null;
8
9       // Getter für die Einstellungen
10      Public.allSettings getter = function() {
11          return _mySettings;
12      }
13      Public.setting = function(theName) {
14          return _mySettings[theName];
15      }
16
17      Base.realize = Public.realize;
18      Public.realize = function() {
19          Base.realize();
20          // initiale Einstellungen festlegen
21          _mySettings = {
22              "position": {"lon": 13.4144, "lat": 52.5235, "zoom": 12},
23              "tilespath": "CONTENT/tiles/"
24          };
25          // Position und Hintergrundfarbe des Fensters setzen
26          window.position = new Vector2i(0, 140);
27          window.backgroundColor = asColor("252422");
28      };
29      Base.postRealize = Public.postRealize;
30      Public.postRealize = function() {
31          Base.postRealize();
32      };
33  };
```

Listing 5.5: MiamiImpl-Klasse, die das Fenster der Anwendung repräsentiert

All diese in einem Spark-Layout definierten Klassen bedienen sich der Methoden `realize` und `postRealize` und überschrieben diese, um eigene Initialisierungen durchzuführen. `realize` wird zu einem Zeitpunkt ausgeführt, wo die Scene noch nicht erstellt wurde, das heißt: es wurden noch keine Texturen geladen und keine Transformationen durchgeführt. Daher können hier Anweisungen gegeben werden, die keine voll aufgebaute Scene benötigen. Anweisungen in `postRealize` hintgegen können davon ausgehen, dass eine vollständige Scene vorhanden ist, in der sie arbeiten können.

In der Klasse TileGridImpl, deren korresponierende Präsentation nur eine leere, aber positionierte und skalierte Box darstellt, werden zuerst sämtliche Kachelcontainer mit den Daten erstellt, die in der Datei *tile_grid.xml* (siehe Listing 5.2) de-

finiert wurden. Dies geschieht über die Objekte der Klasse *TileContainer* (den Kachelcontainern). Anschließend werden Kacheln (Objekte der Klasse *Tile*) mit den Informationen aus den Kachelcontainern erzeugt und initial mit Bildern belegt. Die Startposition wird dabei im `_mySettings`-Objekt in *MiamiImpl.js* festgelegt. Nach Anlegen von EventListenern für Touch- und Mausinteraktion wird die gesamte Scene dargestellt. Der Nutzer kann nun mit der Karte interagieren.

6 Demonstration und Auswertung

In diesem Kapitel soll die Funktionalität des entwickelten Prototypen demonstriert und hinsichtlich der Anforderungen aus Kapitel 3 ausgewertet werden. Weiterhin werden auftretende Probleme beschrieben und versucht Lösungsansätze zu geben.

6.1 Demonstration der Funktionalität

Im Folgenden wird die Nutzung und Funktionalität des entwickelten Systems demonstriert.

Nachdem die Anwendung gestartet wurde, wird dem Nutzer die Karte mit einer im Quelltext voreingestellten Startposition angezeigt. Wie man in Abbildung 6.1 sehen kann, befinden sich auch außerhalb des sichtbaren Bereichs (schwarz umrandet) bereits vorgeladene Kartenkacheln.

Der Nutzer kann nun entweder per Maus oder Berührung mit der Karte interagieren. Zum Verschieben der Karte per Berührung muss man den Tisch wie ebenfalls in Abbildung 6.1 dargestellt mit einer Hand berühren und dann über die Oberfläche wischen. Um die Karte mit der Maus zu verschieben, genügt simples Drag'n'Drop. Wird die Karte über eine gewisse Distanz hinaus verschoben, werden Kartenkacheln nachgeladen.

Weiterhin ist es möglich in die Karte hinein- oder aus ihr herauszuzoomen. Dies geschieht per Berührung mit einer Multi-Touch-Geste. Zum Hineinzoomen müssen sich zwei Touch-Cursor voneinanderweg, zum Herauszoomen zueinanderhin bewegen. Bei der Maus übernimmt diese Funktionalität das Scrollrad.

6.2 Auswertung

Hinsichtlich der Kartendarstellung erfüllt der Prototyp die grundlegensten und damit wichtigsten Anforderungen. Die Kacheln werden in ihrer originalen Größe dargestellt, wodurch eine höchstmögliche Auflösung gewährleistet wird. Außerdem

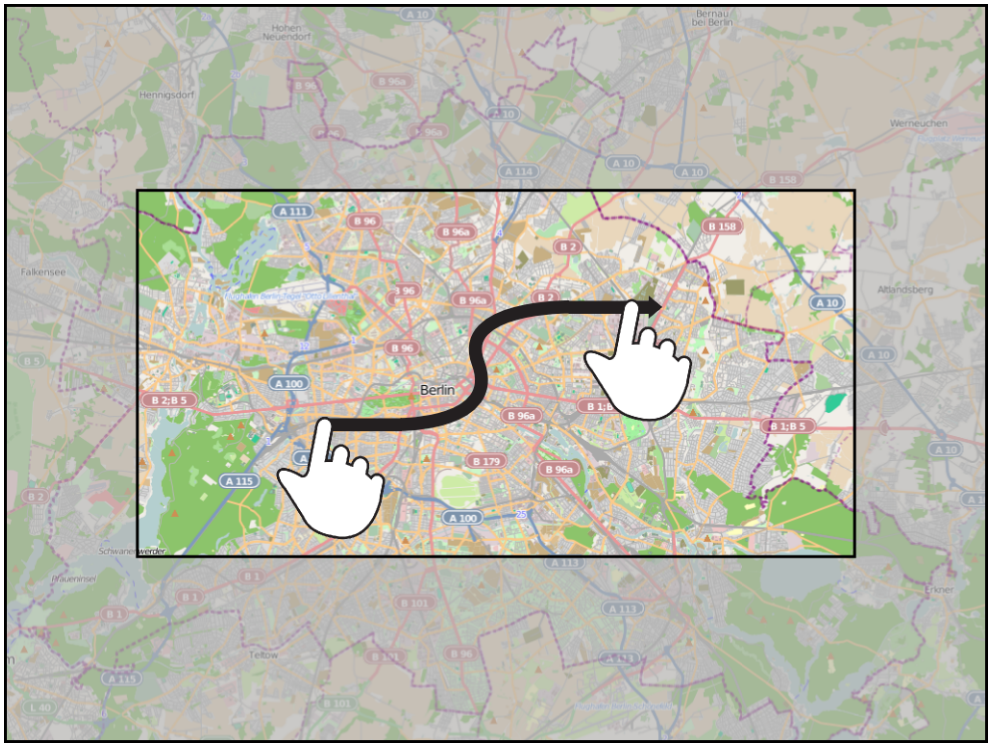

Abbildung 6.1: Screenshot des Prototypen mit schematischen Verdeutlichungen

wird durch das Vorladen noch nicht sichtbarer Kacheln eine flüssige Darstellung der Karte bei Interaktionen sichergestellt.

Die Darstellung einer Legende oder verschiedener Ebenen zur Informationsanreicherung wurde in der Bearbeitungszeit nicht implementiert.

Das Verschieben und Zoomen der Karte ist grundsätzlich möglich, obwohl sich bei Tests der Anwendung bezüglich der Stabilität zwei Mängel feststellen ließen, einer davon gravierend. Beide haben mit der zuverlässigen Darstellung der Karte bei Interaktion zu tun. Zum einen stürzt die Anwendung ab, wenn auf eine nicht vorhandene Stufe gezoomt wird. Dies lässt sich allerdings mit einer Prüfung auf vorhandene Zoomstufen und dementsprechenden Zulassen oder Blockieren der Zoom-Interaktionen beheben. Gravierender hingegen ist das scheinbar willkürliche Auftreten eines Fehlers, der die Karte sprungweise aus dem sichtbaren Bereich schiebt. Die Ursache dieses Fehlers konnte innerhalb des Bearbeitungszeitraumes nicht festgestellt werden.

7 Zusammenfassung und Ausblick

Dieses Kapitel stellt die Zielsetzung dieser Arbeit, die Anforderung an den Prototypen und deren Umsetzung gegenüber und zeigt mögliche Potentiale und Erweierungen auf.

7.1 Geschaffene Lösung

Ziel der Arbeit war es eine Anwendung zur interaktiven Nutzung von geografischen Daten unter Verwendung der Multi-Touch-Technologie zu erstellen.

Der geschaffene Prototyp bedient sich der Daten des freien Projektes OpenStreetMap, einem gemeinschaftsgetriebenen Geoinformationssystem, und des ebenfalls freien Mapnik-Toolkits zum Rendern von Karten, um eine hochdetailierte Karte zu generieren, mit der sowohl per Multi-Touch als auch per Maus interagiert werden kann. Die Anwendung läuft auf einer großformatigen Multi-Touch-Tabletop-Installation, welche ein kapazitives Verfahren zur Erkennung von Berührungen nutzt.

Implementiert wurden im Rahmen dieser Arbeit die Grundfunktionalitäten Darstellung, Verschieben und Zoomen der Karte.

7.2 Potentiale und Erweiterungen

Neben den Möglichkeiten, die mit den Anforderungen beschrieben wurden, bietet das System noch weit mehr Raum für Erweiterungen, um die Funktionalität zu erhöhen.

Eine davon wäre eine tiefgehende Intergration in die OpenStreetMap-Infrastruktur, so dass zum Beispiel geografische Kartendaten direkt per Touch-Eingaben geändert werden können.

Eine andere Erweiterung wäre die Kombination des Systems mit einem Kommunikationsprotokoll. So könnten Daten, die auf dem stationären Gerät übersichtlich

dargestellt und manipuliert werden, an andere, besonders mobile, Endgeräte weitergegeben werden. Dadurch würde ein hoch effizientes Werkzeug zur Koordination in verschiedensten Szenarien enststehen. Die Anwendung würde dabei von groß angelegten Veranstaltungen wie Messen bis hin zu Hilfseinsätzen in Katastrophengebieten reichen.

Quellen

[1] de Lange, N.: Geoinformatik in Theorie und Praxis. Springer, 2005. - ISBN 3540282912

[2] Crockford, D.: JavaScript: The Good Parts. O'Reilly, 2008. - ISBN 978-0-596-51774-8

[3] iPhone Human Interface Guidelines. 2010. Zugriff am 06.06.2010: http://developer.apple.com/iphone/library/documentation/userexperience/conceptual/mobilehig/MobileHIG.pdf

[4] Han J. Y. Multi-Touch Sensing through Frustrated Total Internal Reflection. 2005. Zugriff am 06.06.2010: http://cs.nyu.edu/~jhan/ftirsense/

[5] Villamor, C.; Willis, D.; Wroblewski, L.: Touch Gesture Reference Guide. 2010. Zugriff am 06.06.2010: http://www.lukew.com/touch/

[6] Y60 Platform - Source Code Overview - ART+COM Technologies. Zugriff am 06.06.2010: http://y60.artcom.de/redmine/wiki/y60/Source_Code_Overview

[7] Comparing Touchscreen Technologies - Mass Multimedia, Inc. 2009. Zugriff am 06.06.2010: http://www.touchscreens.com/intro-touchtypes.html und http://www.touchscreens.com/intro-touchtypes-saw.html

[8] Industry Focus: Touchscreens Press Deep Into Consumer Electronics - Keuling, C. - ECN Magazine 2008. Zugriff am 06.06.2010: http://www.ecnmag.com/Articles/2008/11/Industry-Focus–Touchscreens-Press-Deep-Into-Consumer-Electronics/

[9] Jedes fünfte Handy 2009 mit Touchscreen - Fügemann, F. - pressetext Nachrichtenagentur GmbH 2009. Zugriff am 06.06.2010: http://pressetext.de/news/090417001/jedes-fuenfte-handy-2009-mit-touchscreen/

[10] OpenStreetMap. Letzter Zugriff am 06.06.2010: http://wiki.openstreetmap.org/ und http://www.openstreetmap.org/

[11] PostGIS - Refractions Research. Letzter Zugriff am 07.06.2010: http://postgis.refractions.net/

[12] So funktioniert ein Touchscreen - Rügheimer, H. - connect 2009. Zugriff am 09.06.2010: http://www.connect.de/themen_spezial/Kapazitiv-Display-als-Kondensator_5785724.html und http://www.connect.de/themen_spezial/Kapazitiv-Display-als-Kondensator_5785724.html

[13] Google Maps. Letzter Zugriff am 09.06.2010: http://maps.google.de/

[14] Bing Maps. Letzter Zugriff am 09.06.2010: http://www.bing.com/maps/

[15] ART+COM Technologies. Letzter Zugriff am 09.06.2010: http://tech.artcom.de/de/products/touchmaster und http://tech.artcom.de/de/products/y60

Abbildungsverzeichnis

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig und unter ausschließlicher Verwendung der angegebenen Quellen und Hilfsmittel erstellt habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, der xx Juni 2010

_____

Max Ludwig