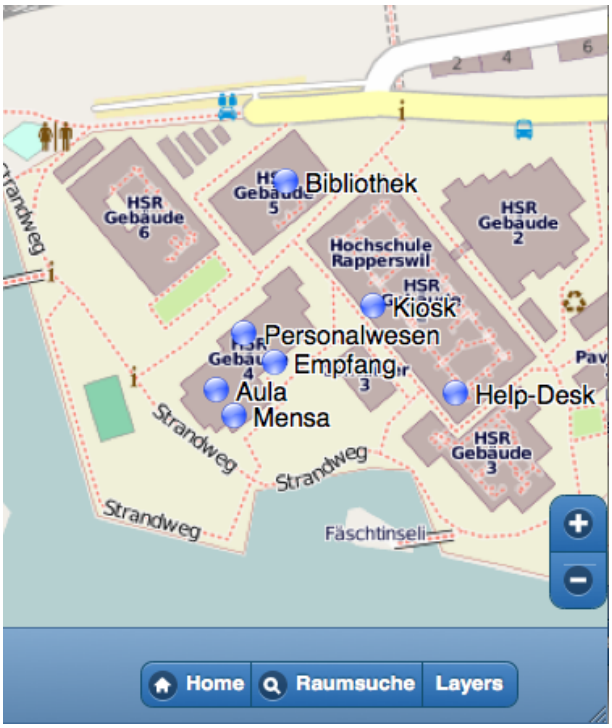
EVALUATION VON OPENLAYERS MOBILE & KHTMLIB SOWIE ERWEITERUNG DER KHTMLIB AUF BASIS VON HTML5



Bachelorarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Frühjahrssemester 2011

Autoren: Florian Hengartner, Stefan Kemper
Betreuer: Prof. Stefan F. Keller
Projektpartner: Bernhard Zwischenbrugger, Wien
Experte: Claude Eisenhut
Gegenleser: Prof. Hansjörg Huser

Stefan Kemper und Florian Hengartner: *Evaluation von OpenLayers Mobile & khtmlib sowie Erweiterung der Khtmlib auf Basis von HTML5,* Bachelorarbeit, © Juni 2011

ZUSAMMENFASSUNG

Khtmlib ist eine JavaScript-Bibliothek zur Darstellung von Webkarten auf mobilen Endgeräten. Im Gegensatz zum weit verbreiteten OpenLayers, ist Khtmlib von Grund auf neu und als leichtgewichtige Alternative dazu entworfen worden. Dabei wird der HTML5-Standard konsequent ausgenutzt. Die vorliegende Arbeit verfolgte das Ziel den Code von Khtmlib durch Refactoring zu verbessern und um verschiedene Funktionalitäten zu erweitern.

Dabei wurde zuerst OpenLayers evaluiert und mit dem Funktionsumfang von Khtmlib verglichen. Als entsprechende Einarbeitung in OpenLayers bot sich ein Refactoring des interaktiven Lageplans der Hochschule für Technik Rapperswil (HSR) an.

Als erste Erweiterung von Khtmlib bot sich das Vektorformat KML an, welches durch Google Earth bekannt geworden ist. Zu den weiteren Verbesserungen an Khtmlib gehören die sog. Groundoverlays (Rasterbilder) - inklusive rotation des Bildes, die Möglichkeit Elemente in Layern zu gruppieren und verschachteln, das Einbinden von WMS, die Vereinheitlichung des Renderings von sog. Overlays und die Adaption des Klassenmodells der Vektor Klassen auf den Simple Feature Access Standard. Die verwendeten JavaScript-Bibliotheken sind jQuery, das Plugin jQueryRotate und für das Unit Testing wurde Jasmine eingesetzt. Die Dokumentation wurde entsprechend angepasst und mit Demobeispielen ergänzt.

Die erweiterte und getestete Khtmlib-Bibliothek wird nach Abschluss der Arbeit dem Erstautor übergeben und soll in das Repository integriert werden. Für weitere Infos siehe `http://wiki.hsr.ch/StefanKeller/SKemperFHengartnerAufgabenstellung`.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

ACRONYMS

API Application Programming Interface

BA Bachelorarbeit

DRY Don't Repeat Yourself

GIS Geographic Information System

GUI Graphical User Interface

HSR Hochschule für Technik Rapperswil

JSON Javascript Object Notation

KML Keyhole Markup Language

OGC Open Geospatial Consortium

OSM OpenStreetMap

POI Point Of Interest

RIA Rich Client Applications

SVN Subversion

UI User Interface

W3C World Wide Web Consortium

YUI Yahoo! User Interface

Zusätzliche und ausführlichere Erklärungen befinden sich im Glossar unter Anhang B (S. 127)

Teil I

ÜBERSICHT

1 AUFGABENSTELLUNG

Der Titel der Bachelorarbeit (BA) im Frühjahrssemester 2011 von Stefan Kemper und Florian Hengartner der Abteilung Informatik lautet "khtmlib_hsr — Evaluation von OpenLayers Mobile und khtmlib sowie Erweiterung der khtmlib auf Basis von HTML5"

1.1 AUSGANGSLAGE

Mit HTML5 zeichnet sich ein neuer Standard ab, der u.a. die Entwicklung von Rich Client Applications (RIA) auf Smartphones und Tablets vereinfachen soll. Mit OpenLayers gibt es einen Quasi-Standard zur Darstellung von Karten in Desktop-Browser. Zur Beherrschung dieser Technologien sollen Erfahrungen gesammelt werden.

Ein typischer Use-Case ist die Anzeige einer Webkarte mit Point Of Interests (POIs), gesteuert mit Touchscreen-Gesten.

1.2 AUFGABENSTELLUNG

Als Client-Library steht die Erweiterung von khtmlib im Vordergrund. Serverseitig wird dazu das eigene Projekt OpenStreetMap-in-a-Box oder ein ähnlicher POI-Server vorgegeben.

Am Schluss soll eine Software resultieren, die auch ausserhalb des eigenen Bachelorarbeit-Settings demonstrierbar ist — beispielsweise eine Mobile-Client-Version um Webkarten ("Switzerland delivered by OpenStreetMap-in-a-Box") anzuschauen, sowie ein robuster Beitrag zu Khtmlib.

1.2.1 Mögliche Aufgaben

- Evaluieren OpenLayers Mobile
- Demo-Website auf Basis Khtmlib
- Evaluieren weitere JavaScript-Bibliotheken (GeoExt, etc.)
- Ausbau der Funktionalitäten von Khtmlib

1.2.2 Funktionen Khtmlib

- Support von WMS-Tiles aus GeoWebCache analog OpenLayers
- Ausbau (inkl. Unit-Tests) der Vektor-Klassen "Low Level"
- Ausbau Vektor-Klassen zu Styling
- KML darstellen

1.3 VORGABEN

- Die Sprache ist englisch in Code, Installationsanleitung, allfälliger Benutzerdokumentation, Präsentationsfolien. Sonst deutsch.
- Software:
  - Client: Khtmlib, OpenLayers, ein JUnit-Framework.
  - Server: keine

1.4 BETEILIGTE

1.4.1 Diplomanden

Siehe oben.

1.4.2 Projektpartner

GISpunkt HSR und Open Source Community.

1.4.3 Betreuung HSR

Verantwortlicher Dozent: Prof. Stefan Keller, IFS-HSR (sfkeller@hsr.ch)
Gegenleser: Prof. Hansjörg Huser, IFS-HSR
Experte: Claude Eisenhut, Eisenhut Informatik Burgdorf (ceisenhut@eisenhutinformatik.ch)

1.5 PROJEKTABWICKLUNG

1.5.1 Termine

- Beginn der Arbeit: Semesterbeginn.

- Abgabetermin: zwei Wochen nach Semesterende, 12h.
- HSR-Forum sowie weitere Termine: Siehe `https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html` (intern).

1.5.2 Arbeitsaufwand

Für die erfolgreich abgeschlossene Arbeit werden 12 ECTS angerechnet. Dies entspricht einer Arbeitsleistung von 360 Stunden.

1.5.3 Hinweise für die Gliederung und Abwicklung des Projektes

Gliedern Sie Ihre Arbeit in vier bis fünf Teilschritte. Schliessen Sie jeden Teilschritt mit einem Meilenstein ab. Definieren Sie für jeden Meilenstein, welche Resultate dann vorliegen müssen!

1.5.3.1 *Folgende Teilschritte bzw. Meilensteine sollten in der Planung vorgesehen werden:*

- Schritt 1: Projektauftrag inkl. Projektplan (mit Meilensteinen)
- Meilenstein 1: Review des Projektauftrages abgeschlossen. Projektauftrag von Auftraggeber und Dozent genehmigt Termin: ca. zwei Wochen nach Beginn der Arbeit
- Letzter Meilenstein: Systemtests abgeschlossen. Termin: ca. eine Woche vor Abgabe
- Die Software ist in einem agilen, d.h. iterativen und inkrementellen Prozess zu entwickeln: Man plane möglichst früh einen ersten lauffähigen Prototypen mit den wichtigsten und kritischsten Kernfunktionen. In den folgenden Phasen kann dann schrittweise ausgebaut und getestet werden.
- Falls in der Arbeit neue oder unbekannte Technologien eingesetzt werden, sollte man parallel zum Erarbeiten des Projektauftrages mit dem Technologiestudium beginnen.
- Es sind Unit-Tests einzusetzen. Software und Dokumente werden auf einem Repository verwaltet (z.B. SVN, git). Evt. ist ein Build Server hilfreich (z.B. Cruise Control).
- Man halte sich im Übrigen an die Vorgaben aus dem Modul SE-Projekt.

1.5.3.2 *Projektadministration:*

- Es ist ein Projekttagebuch zu führen aus dem ersichtlich wird, welche Arbeiten durchgeführt wurden (inkl. ungefährem Zeitaufwand). Diese Angaben sollten ggf. eine individuelle Beurteilung ermöglichen.
- Die Arbeiten sind laufend zu dokumentieren. Man lege die Projektdokumentation mit der aktuellen Planung und den Beschreibungen der Arbeitsresultate elektronisch in einem Projektordner ab. Dieser Projektordner sollte jederzeit einsehbar sein (z.B SVN-Server oder File-Share).

1.5.3.3 *Fortschrittsbesprechung:*

- Regelmässig findet zu einem fixen Zeitpunkt eine Fortschrittsbesprechung statt (Arbeiten im Ausland: Wochenbericht).
- Teilnehmer sind Dozent und Studenten, bei Bedarf auch Vertreter der Auftraggeber
- Termin gem. Absprache (Arbeiten im Ausland z.B. Montags).
- Falls notwendig, können weitere Besprechungen / Diskussionen einberufen werden.
- Sie erstellen zu jeder Besprechung ein Kurzprotokoll, welches Sie spätestens 2-3 Tage nach der Sitzung per E-Mail an den Betreuer senden (Arbeiten im Ausland: Eintrag im Wiki o.ä.).

1.5.4 Inhalt der Dokumentation

- Die fertige Arbeit muss folgende Inhalte haben:
  1. Abstract, Management Summary, Aufgabenstellung
  2. Technischer Bericht
  3. Dokumente der Projektdokumentation
  4. Anhänge (Literaturverzeichnis, CD-Inhalt)
- Mind. die Dokumente der Punkte 1, 2 sowie Installation und Code müssen in Englisch sein.
- Die Abgabe ist so zu gliedern, dass die obigen Inhalte klar erkenntlich und auffindbar sind.
- Zitate sind zu kennzeichnen, die Quelle ist anzugeben.

- Verwendete Dokumente und Literatur sind in einem Literaturverzeichnis aufzuführen.
- Projekttagebuch, Dokumentation des Projektverlaufes, Planung etc.
- Weitere Dokumente (z.B. Kurzbeschreibung für Broschüre, Poster) gemäss www.hsr.ch und gemäss Absprache mit dem Betreuer.

1.5.5 Form der Dokumentation

- Bericht (Struktur gemäss Beschreibung) gebunden (2 Exemplare) und in Ordner (1 Exemplar „kopierfähig“ in losen, gelochten Blättern).
- Alle Dokumente und Quellen der erstellten Software auf CD; CD's sauber angeschrieben (3 Ex.).

1.5.6 Bewertungsschema

1.5.6.1 *Als Bewertungsschema gilt das in www.hsr.ch erwähnte, d.h. die folgenden Aspekte werden bewertet:*

- Projektorganisation (Gewicht 1/6)
- Bericht (Gewicht 1/6): Inhalt, Gliederung, Sprache
- Inhalt (Gewicht ½): 1. Vorstudie, Anforderungsanalyse und Domainanalyse; 2. Entwurf (Systemarchitektur, Beschreibung des Entwurfs, Entwurf Benutzerschnittstelle); 3. Realisierung und Test
- Mündliche Prüfung (Gewicht 1/6)

Es gelten ansonsten die üblichen Regelungen zum Ablauf und zur Bewertung der BA-Arbeit des Studiengangs Informatik der HSR.

Rapperswil, 2. März 2010, S. Keller

2 MANAGEMENT SUMMARY

2.1 AUSGANGSLAGE

Bis eine Karte auf einem Browser dargestellt wird, sind viele Arbeitsschritte involviert. Es fängt mit dem sammeln und aufbereiten der Daten an und endet mit der Betrachtung eines Nutzers auf einem Endgerät. Im Bereich der Geographic Information System (GIS) existieren verschiedene Datenformate und diverse Arten, diese Daten an den Benutzer auszuliefern. Für die Darstellung der Daten auf dem Endgerät existieren verschiedene Programme/Bibliotheken, die in den ganzen Prozess involviert sind.

Unsere Arbeit befasst sich mit den Bibliotheken welche für die Darstellung der Karten im Browser und speziell für mobile Endgeräte (Smartphones/Tablets) entwickelt wurden.

Zur Darstellung von Karten auf Browsern können verschiedene Javascript-Bibliotheken verwendet werden. Jeder Kartenanbieter wie Google Maps, Yahoo! Maps oder Bing Map stellt eine eigene Bibliothek bereit. Im Opensource Bereich gibt es OpenLayers und Khtmlib. OpenLayers glänzt durch einen extremen Funktionsumfang der sehr viele Bereiche abdeckt. Sie ist die Standard-Bibliothek zur Darstellung von OpenStreetMap (OSM) Karten. Neben dieser ist sie auch fähig diverse weitere Formate und Application Programming Interface (API)'s (z.B. Google Maps) anzusteuern. Die Khtmlib Bibliothek kann unter anderem OSM Karten darstellen und hat einen sehr schlanken Umfang.

Khtmlib hat die bessere Unterstützung von mobilen Endgeräten als OpenLayers. Unter anderem unterstützt es panning[1] auf Geräten mit Touch-Screen, stufenloser Zoom, 3DCSS für Hardwarerendering (auf Browsern die dies unterstützen), für mobile Geräte optimierte Performance, usw..

2.2 ZIELSETZUNG

Die beiden Bibliotheken OpenLayers und Khtmlib sollen im Bereich der mobilen Endgeräten einander gegenüber gestellt werden.

1 Verschieben der Karte - Glossar Abschnitt B.14, S. 129

Für die Einarbeitung in GIS und OpenLayers wurde der Interaktive Lageplan [2] der HSR ausgebaut.
Eine grobe Liste der Verbesserungen:

- Überarbeitung des User Interface (UI)
- Verbesserung der Suche: Raumposition markieren
- Verwendung der neusten OpenLayers Version
- Für jedes Gebäude alle Stockwerkpläne anzeigen
- Permalink

Der Schwerpunkt dieser Arbeit ist die Verbesserung und Erweiterung der Khtmlib-Funktionalität.

- Unterstützung eines Subsets von KML
- Erstellen der für den KML Support benötigten neuen Features:
  - GroundOverlays: Rasterbilder auf Karte einblenden
  - Layers: Container für Elemente auf der Karte
  - Überarbeitung aller Elemente zur Kompatibilität mit Layers
- Unittests für bestehende Codebasis
- Refactoring des Codes zur Verbesserung der Wartbarkeit und Erweiterbarkeit

2.3 VORGEHEN

Für die Einarbeitung in OpenLayers bot sich das Refactoring des interaktiven Lageplans der HSR an. Dies bot einen guten Einstig in die GIS Welt. Als nächster Schritt wurde der Funktionsumfang von OpenLayers mit demjenigen von Khtmlib verglichen. Dabei interessierte vor allem der Bereich mobile Endgeräte sprich Smartphones und Tablets.

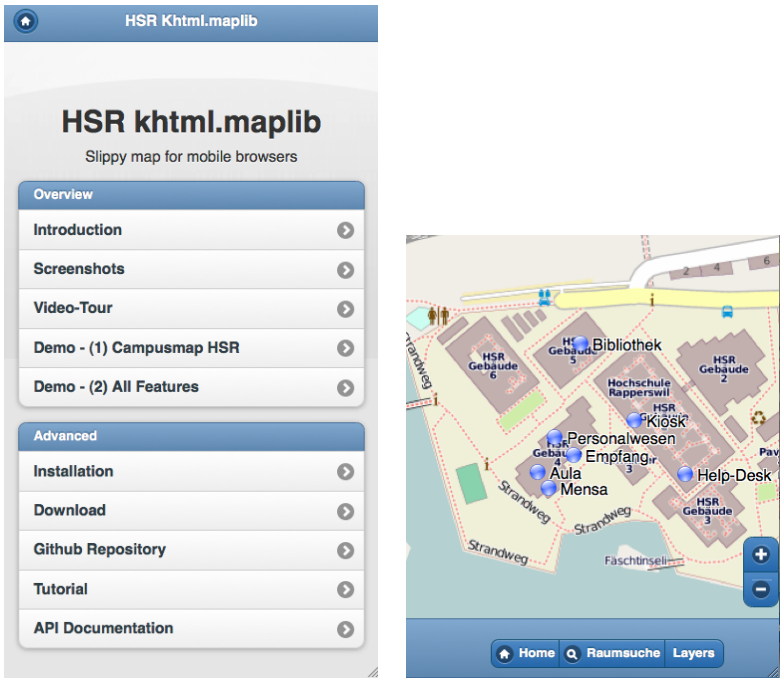
Nach Abschluss der Verbesserungsarbeiten am Lageplan entschlossen wir uns den Support von KML in Khtmlib zu implementieren. KML ist ein Vektorformat welches durch Google Earth bekannt wurde. Es erlaubt den Austausch von einfachen geometrischen Daten bis hin zu Kamerafahrten. Das Format beinhaltet nebst den Daten auch Anweisungen zur Darstellung (Styling).

2 http://labs.geometa.info/campusmap/

Als nächste Erweiterung wurden GroundOverlays eingebaut. Dies erlaubt das Einblenden von Rasterbildern auf der Karte, inkl. Rotation der Bilder.

Während den ersten Arbeiten an Khtmlib wurde klar, dass ein Refactoring nötig ist. Die "neue" Codebasis mit kleineren Bausteinen, geringeren Abhängigkeiten und klaren Verantwortlichkeiten hat sich bei der Erstellung von neuen Features gelohnt.

Zuletzt wurde eine Webseite konzipiert für Mobile-Browser als Showcase erstellt, welche den Einsatz von Khtmlib demonstriert und Informationen über das Projekt bereitstellt.



(a) Menu **(b)** Lageplan

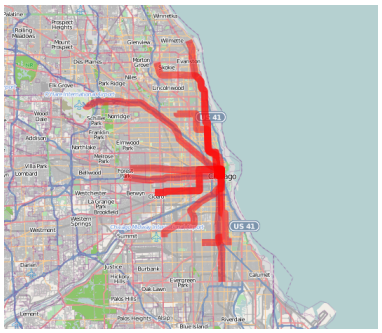
Abbildung 2.1: Showcase

2.4 ERGEBNIS

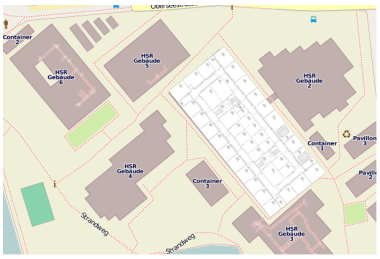
Folgende Resultate wurden erreicht:

- Verbesserung des interaktiven Lageplans
- Khtmlib
  - Unterstützung eines Subsets von KML (einlesen und darstellen)
  - GroundOverlays (Rasterbilder)

- Layer (Container/Gruppierung von Elementen auf der Karte)
- WMS Karten anziehen
- Refactoring der Codebasis
- Verbesserung der Stabilität der Khtmlib Karte durch diverse Bugfixes
- Funktionierendes Vektor-Rendering auf Android Browser
- minimaler Support für die Navigation auf 'Mobile Internet Explorer'

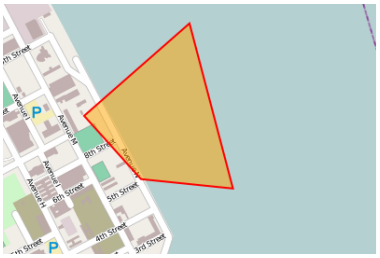


(a) Chicago Metro Linien in Rot

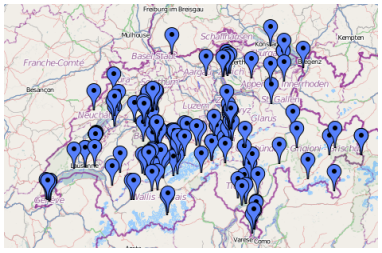


(b) Gebäudeplan als GroundOverlay

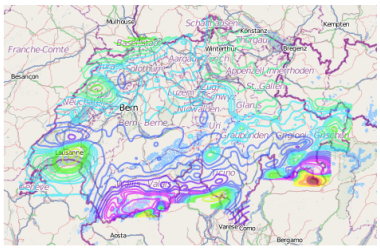
(c) Polygone



(d) LinearRing



(e) Placemarks



(f) WMS (Aeromagnetik)

Abbildung 2.2: Khtmlib Beispiele (Daten für a–e aus KML-Datei)

Damit bestehende Codebeispiele und manuelle Tests nicht verloren gehen wurden sie trotz grossem zeitlichem Aufwand an die neue API angepasst.

Bei der Erstellung der Unittests und Benutzerdokumentation wurde Wert darauf gelegt, nicht nur die in dieser Arbeit erstellten Features zu testen und dokumentieren. Als Beitrag ans Khtmlib Projekt wurde bereits Bestehendes in die Dokumentation und Testsuite aufgenommen.

OPENLAYERS MOBILE CODE-SPRINT Kurz vor bzw. während unserer Arbeit wurde in einem Code-Sprint der Support für mobile Endgeräte in der OpenLayers- Bibliothek implementiert. Dank diesen Erweiterungen kann man sagen, dass OpenLayers ebenso geeignet ist für den Einsatz auf mobilen Endgeräten.

Teil II

TECHNISCHER BERICHT

3 EINFÜHRUNG

3.1 PROBLEMSTELLUNG, VISION

3.1.1 Interaktiver Lageplan

Der bestehende interaktive Lageplan ist eine Karte des HSR Campus. Zur Implementation wurde OpenLayers verwendet. Beim Lageplan soll neu die aktuellste Version von OpenLayers verwendet werden. Anhand einiger sinnvollen Verbesserungen bei der Raumsuche, Permalink, sowie dem Anzeigen der Stockwerkpläne sollen sich die Studenten mit OpenLayers vertraut machen.

3.1.2 Erweiterung Khtmlib

Anwender von Khtmlib haben eine für mobile Endgeräte optimierte Software. Der Funktionsumfang von Khtmlib ist jedoch recht bescheiden. Um diesen Benutzern weitere Anwendungsmöglichkeiten zu geben, soll der Funktionsumfang erweitert werden.

3.2 ZIELE

3.2.1 Interaktiver Lageplan

Zu den bestehenden Features gehört eine Raumsuche und die Anzeige verschiedener POIs (Mensa, Aula, usw.). Die verwendete OpenLayers Version im interaktiven Lageplan ist veraltet und soll aktualisiert werden. Bei der bestehenden Raumsuche wird als Resultat das Gebäude in dem sich der Raum befindet auf der Karte hervorgehoben. Neu soll die Position des gefundenen Raumes auf der Karte angezeigt werden. Pro Stockwerk (Erdgeschoss, 1. Stock, 2. Stock) soll ein Layer mit dem jeweiligen Stockwerkplan eingeblendet werden. Der Stockwerkplan soll über das betreffende Gebäude gelegt werden. Aktuell geht beim versenden eines Links zur Karte die Einstellungen (Zoomstufe, Position, Suchbegriff, aktive Layer) verloren. Um dies zu verbessern

soll ein Permalink implementiert werden, der alle Kartenparameter übernimmt.

3.2.2 Erweiterung Khtmlib

Khtmlib soll um neue Funktionalität erweitert werden. Im wesentlichen sind dies ein KML-Parser, Anpassung der Vektor-Funktionalität, GroundOverlays, Layers, WMS-Overlay, das Beheben von Bugs, Navigiermöglichkeit für Windows-Mobile-Browser und das Erstellen von Unittests auch für bereits vorhandenen Code. Ausserdem soll die fehlende API Dokumentation für das gesamte Projekt erstellt und das Benutzerhandbuch verbessert werden.

3.3 RAHMENBEDINGUNGEN

Die Rahmenbedingungen sind durch die Aufgabenstellung (siehe Kapitel 1) sowie durch die von der HSR definierten Termine für Abgabe und Präsentation gegeben.

3.4 VORGEHEN

Das generelle Vorgehen ist im Management Summary (siehe Abschnitt 2.3) beschrieben. Im Folgenden wird aufgezeigt, wie das Vorgehen im Vorprojekt sowie im Hauptprojekt im Detail war.

3.4.1 Interaktiver Lageplan

In einem ersten Schritt wurde der bestehende Code analysiert. Anschliessend wurden zusammengehörende Funktionen gruppiert sowie gewisse bestehende Fehler behoben.

In einem zweiten Schritt wurden die neu gewünschten Features (Permalink, Suche, auf Suchresultate zoomen und markieren der Resultate) implementiert.

In einem letzten Schritt wurde das Graphical User Interface (GUI) den Anforderungen der HSR angepasst, sowie Schlussfolgerungen aus dem Gelernten für die Erweiterungen an Khtmlib gezogen.

3.4.2 Erweiterung Khtmlib

In Zusammenarbeit mit dem Maintainer (Bernhard Zwischenbrugger) der Khtmlib Bibliothek wurde eine Liste der möglichen Erweiterungen aufgestellt. Diese wurde zusammen mit dem Betreuer priorisiert und abgearbeitet.

Während der ganzen Entwicklungsphase wurde Bernhard Zwischenbrugger regelmässig über den aktuellen Arbeitsstand informiert. Er hatte die Möglichkeit seine Wünsche und Vorstellungen einzubringen.

Zu Beginn wurden Unittests für den bestehenden Code geschrieben. Damit verbunden war auch die Einarbeitung in den bestehenden Code. Alle Entwicklungen wurden nach dem Prinzip "test first" entwickelt.

Die ursprüngliche Idee, Styling mit MapCSS möglich zu machen, wurde zugunsten von der Entwicklung eines KML-Parsers aufgegeben. Dabei wurde in einer ersten Phase ein Parser gebaut, der auf die Hierarchie von KML-Dateien keine Rücksicht nahm. Für die Abbildung der Daten aus dem KML in Objekte wurde es nötig weitere Klassen in Khtmlib zu erstellen.

In einer zweiten Phase wurde der Parser so erweitert, dass auch die Hierarchien in Objekten abgebildet werden konnten. Dafür wurde die Klasse Layer entwickelt. Diese kann ihrerseits wieder Layer-Objekte enthalten. Dadurch können die Hierarchien abgebildet werden.

Parallel zur Implementation des Parsers wurde der bestehende Code refactored. Dabei wurden grosse Klassen aufgeteilt in kleinere weniger umfangreiche Klassen. Vererbung wurde über das Mixin-Pattern eingeführt und Geometrie-Klassen nach dem der Geometrie-Typen-Hierarchie des Standard SFS[1] eingeführt.

In der letzten Phase wurden weniger aufwendige Features entwickelt (WMS Overlay anbinden, minimales navigieren auf Windows Mobile Browsern oder Zoomen auf eingelesene KML-Daten) und Bugs korrigiert sowie eine Demoapplikation erstellt um die neuen Features Präsentieren zu können.

Parallel zum ganzen Projekt wurde die Dokumentation der Bibliothek weiter ausgebaut und ein Tutorial erstellt um sicher zu gehen, dass der Einsatz der Bibliothek für den Endbenutzer so einfach und verständlich wie möglich ist.

1 http://www.opengeospatial.org/standards/sfs

4 STAND DER TECHNIK

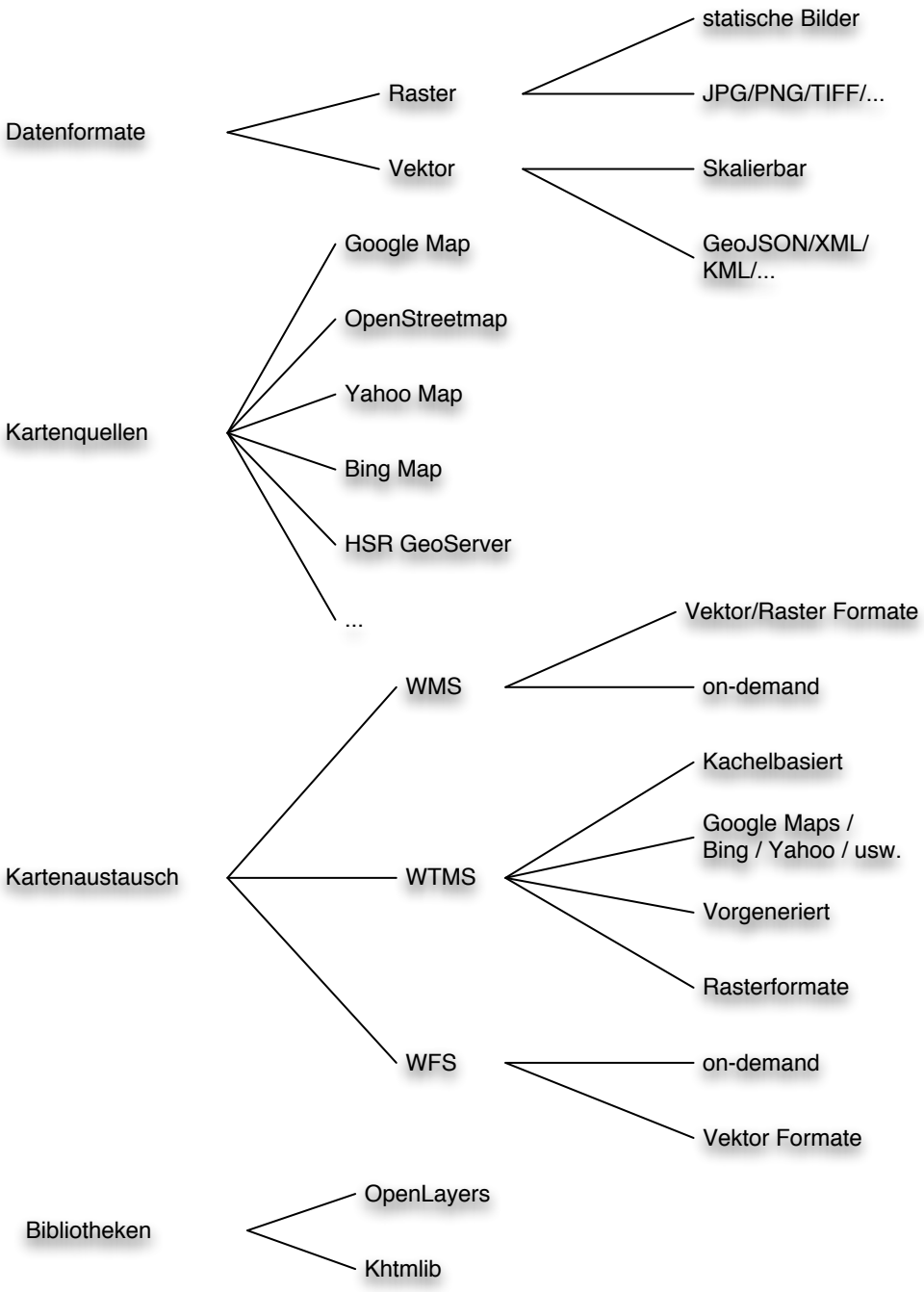
Für die Darstellung von Karten in Webseiten existieren bereits einige Javascript-Bibliotheken. Der Funktionsumfang innerhalb der Bibliotheken variiert von Bibliothek zu Bibliothek stark.

Neben den kartenspezifischen Bibliotheken gibt es auch viele allgemeine Javascript-Bibliotheken. Dabei haben sich in der letzten Zeit einige hervorgetan, die für den Einsatz mit mobilen Endgeräte optimiert sind.

Im Folgenden werden die wichtigsten Vertreter kartenspezifischer und allgemeiner Javascript-Bibliotheken erläutert und deren aktueller Entwicklungsstand aufgezeigt. Ausserdem soll ein Einblick in die GIS Welt gewährt werden.

4.1 EINFÜHRUNG

Abbildung 4.1: Übersicht



4.2 KOORDINATENSYSTEME

Für die vorliegende Arbeit sind vor allem diese Koordinatensysteme wichtig:

EPSG:900913 Google-System (inoffizielle EPSG-Nummer)

EPSG:3857 Google-System (offizielle Nummer)

EPSG:4326 WGS84, GPS-System

EPSG:2056 schweizerisches Koordinatensystem, CH1903+

Details zu den einzelnen Nummern können unter `http://spatialreference.org` nachgeschlagen werden.

Die Koordinatensysteme verfolgen verschiedene Ansätze. Das EPSG:2056 ist ein für die Schweiz optimiertes Koordinatensystem und kommt vor allem in den Schweizer Landeskarten zum Einsatz.

EPSG:4326 ist ein weltweit gültiges Koordinatensystem, welches den Globus in Längen- und Breitengrade einteilt. Es wird bei GPS für die Positionsangabe verwendet.

EPSG:900913 ist das System, welches von Google eingesetzt wird. Es ist optimiert um die Welt in cachebare Kacheln zu unterteilen. [1]

4.3 DATENFORMATE FÜR KARTENINFORMATIONEN

Die verschiedenen spezifischen Bibliotheken können verschiedene Kartendaten anzeigen. Nicht alle Bibliotheken sind für jedes Datenformat geeignet.

4.3.1 KML & KMZ

4.3.1.1 *Einführung*

KML ist ein XML-Basiertes Format zum Austausch von ortsbezogenen Daten und wird beispielsweise von Google Earth verwendet. KMZ ist ein ZIP-Komprimiertes KML-File. [2]

1 Informationen zum Google-System: `http://www.maptiler.org/google-maps-coordinates-tile-bounds-projection/`

2 Quelle: `http://en.wikipedia.org/wiki/Keyhole_Markup_Language`

Die Khtmlib soll KML-Unterstützung erhalten. Die gesamte KML Spezifikation[3] [4] ist sehr gross.

Das spezielle an KML ist, dass es kein reines Daten Format ist. Es werden auch Informationen (die Styles) wie die Daten dargestellt werden sollen mitgeliefert.

4.3.1.2 *KML bei Khtmlib*

Zu Beginn der Arbeit unterstützte Khtmlib KML nicht. Mit unserer Arbeit haben wir KML-Support implementiert (siehe Kapitel 6 und Unterabschnitt 8.2.1).

4.3.1.3 *KML bei OpenLayers*

OpenLayers kann KML lesen und *schreiben*. Von OpenLayers unterstützte KML Tags:

- Link
- NetworkLink
- Style
- StyleMap
- Placemark
- Folder
- MultiGeometry
- Polygon
- LineString
- Point

Quelle: OpenLayers-2.10/lib/OpenLayers/Format/KML.js
API Referenz: `http://dev.openlayers.org/releases/OpenLayers-2.10/doc/apidocs/files/OpenLayers/Format/KML-js.html#OpenLayers.Format.KML.OpenLayers.Format.KML`

3 `http://www.opengeospatial.org/standards/KML/`
4 `http://code.google.com/intl/de/apis/KML/documentation/KMLreference.html`

4.3.1.4 *KML bei Google Maps*

Google Maps unterstützt ein Subset von KML:

- Placemarks
- Icons
- Folders
- Descriptive HTML
  - Entity replacement via <BalloonStyle> and <text>
- KMZ (compressed KML, including attached images)
- Polylines and polygons
- Styles for polylines and polygons, including color, fill, and opacity
- Network links to import data dynamically
- Regions inside network links (for efficient loading of large KML datasets)
- Ground overlays and screen overlays

Quelle: `http://code.google.com/apis/KML/documentation/mapsSupport.html`.

4.3.2 GeoJSON

Bei GeoJSON ist ein Format auf der Basis von Javascript Object Notation (JSON). GeoJSON Daten sind somit Javascript Objekte. In einem Javascript Programm kann sehr einfach aus einem JSON Text, konkrete Objekte erzeugt werden. Für die Darstellung von Daten in JavaScript-Kartenapplikationen ist die Verwendung von GeoJSON deshalb ohne grossen Aufwand zu implementieren.

GeoJSON wird im interaktiven Lageplan der HSR verwendet.

4.3.3 GPX

GPX ist ein Dateiformat, welches für die Speicherung und den Austausch von GPS-Daten entwickelt wurde. GPX-Daten werden als XML abgespeichert.

GPX kann auch von Khtmlib angezeigt werden (siehe Unterabschnitt 4.8.4).

4.3.4 GML

GML steht für Geography Markup Language und ist ein weiteres auf XML basierendes Format um geografische Daten auszutauschen. GML ist ein riesiger von Open Geospatial Consortium (OGC) definierter Standard. Dabei wurden bei GML mehrere Profile definiert (Submengen des ganzen GML-Funktionsumfangs) um die Einführung von GML als Standard zu vereinfachen. [5]

4.4 KARTEN STYLING

Einige Formate (z.B. KML) bringen bereits Styling-Informationen mit, also z.B. welche Farbe eine Linie hat. Bei anderen Formaten (GeoJSON, GPX) fehlt diese Information.

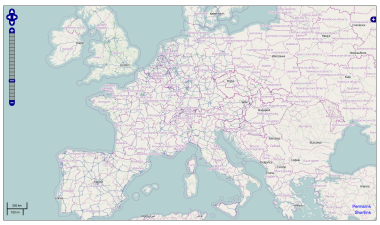
Folgende Formate können für Styling verwendet werden:

STYLED LAYER DESCRIPTOR (SLD) Dieser OGC Standard wird in WMS Systemen verwendet.

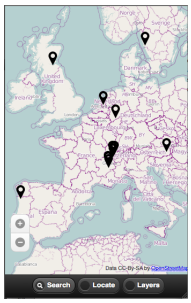
MAPCSS Schlanke Alternative zu SLD. Die Syntax orientiert sich an CSS.

5 Quelle: http://de.wikipedia.org/wiki/Geography_Markup_Language

4.5 JAVASCRIPT-BIBLIOTHEKEN ZUR KARTEN-DARSTELLUNG



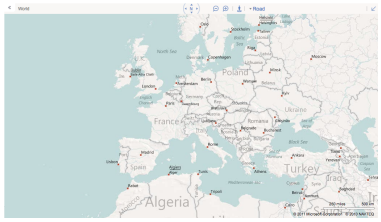
(a) OpenLayers



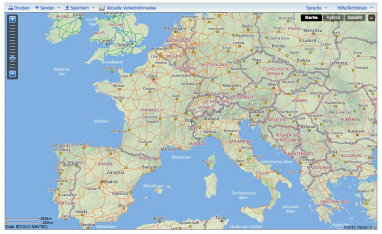
(b) OpenLayers Mobile

(c) Khtmlib

(d) Google Maps



(e) Bing Maps



(f) Yahoo Maps

(g) ArcGIS

Abbildung 4.2: Screenshots

4.5.1 OpenLayers

OpenLayers ist ein OpenSource Projekt. Unterdessen ist es im Web der defakto Standard für die Darstellung von Karten aus dem OpenStreetMap[6] Projekt.

Die Auswahl an Features ist riesig. Es werden Elemente für das GUI mitgeliefert (Zoombar, Permalink, Layer Switcher, usw.), diverse Datenformate unterstützt (GeoJSON, XML, KML, ArcXML, WFS, usw.), diverse geometrischen Datentypen, diverse Layer (GeoRSS, Google, MapServer, WorldWind, Yahoo, WMTS), Vektor Renderer, Konvertierung zwischen den verschiedenen Koordinaten System, und vieles mehr.

4.5.2 Khtmlib

Bei Khtmlib handelt es sich um eine JavaScript-Bibliothek im Entwicklungsstadium. Sie wurde von Grund auf neu gebaut und soll vor allem auf mobilen Endgeräten performant laufen. Die Grundfunktionalität der Karte (panning, zoomen, etc.) existiert bereits. In unserer Arbeit wollen wir diese Bibliothek um weitere Funktionalitäten erweitern.

4.5.3 GeoExt

GeoExt[7] ist ein Plugin für das Javascript Framework ExtJS[8].

ExtJS bietet unter anderem vorgefertigte Widgets zur Gestaltung einer Webapplikation. GeoExt bietet ein solches Widget zur einfachen Einbindung von Karten an.

GeoExt verwendet zur Darstellung von Karten ausschliesslich OpenLayers.

4.5.4 Google Maps API

Die Google Maps API erlaubt das Darstellen von statischen Bildern und interaktiven Karten auf der eigenen Webseite. Ohne eigene Erweiterungen können ausschliesslich Kartenbilder (Kacheln) von Google geladen werden.

Google Maps API bietet viele Möglichkeiten die Karte zu verwenden. Es können Controls (Zoom, Rotate, Pan, ...) und Elemente (Marker, Circle, Polygon, usw.) auf der Karte platziert

6 http://www.openstreetmap.org/
7 http://www.geoext.org/
8 http://www.sencha.com/products/extjs/

werden, geometrische Berechnungen (z.B. Distanz) und weiteres durchgeführt werden.

Vom durch Google Earth verwendeten KML unterstützt die Google Maps API ein Subset[9].

Die Google Maps Terms-Of-Service[10] erlauben, das Einbinden von Google Karten Bilder nur über das Google Maps API. Aus diesem Grund kann Khtmlib offiziell den Basis Layer nicht aus Google Maps Kacheln zusammensetzen obwohl dies technisch über direkten Zugriff auf die Kacheln bereits möglich wäre. Hingegen verwendet OpenLayers das offizielle Google Maps Javascript API zum Darstellen von Google Karten, was deshalb erlaubt ist [11] [12].

4.5.5 ArcGIS Javascript API

ArcGIS bietet ein Paket an GIS Software. Dazu gehört Windows Software zur Darstellung, Bearbeitung und Analyse von Kartenmaterial und Geodatenbank-Serversoftware. Nebst dem Javascript API wird auch eines für Flex und Silverlight zur Verfügung gestellt.

Das Javascript API[13] unterstützt die üblichen Geometrie Typen, diverse GUI Controls und diverse Layers (WMS, OpenStreetMap, ...). Die ArcGIS Javascript API bietet Unterstützung für die Integration von ArcGIS Karten in die Google Maps API's und Bing MAP API's.

Als Basis dient das Javascript Toolkit DOJO[14].

4.5.6 Bing Maps API

Als Konkurrenz zu Google Maps tritt Microsoft mit Bing Maps an.

Über die Bing Maps API[15] können Karten auf der eigenen Webseite eingebunden werden. Es bietet Marker, geometrische Formen, Routing. Im vergleich zu Google Maps und ArcGIS API scheint der Funktionsumfang kleiner zu sein. Es wird standard-

9 http://code.google.com/apis/KML/documentation/mapsSupport.html.
10 http://www.google.com/accounts/TOS
11 http://lists.osgeo.org/pipermail/openlayers-users/2009-June/012225.html
12 http://lists.osgeo.org/pipermail/openlayers-users/2009-June/012209.html
13 http://help.arcgis.com/en/webapi/javascript/arcgis/
14 http://www.dojotoolkit.org/
15 http://www.bingmapsportal.com/isdk/ajaxv7

mässig ebenfalls nur das Anzeigen von Bing Kartendaten ermöglicht.

Für das iPhone gibt es ein Objective-C Control[16] und für das Windows Phone ein Silverlight Control [17]

Ausserdem gibt es ein 'Spatial DATA API' auf REST Basis für die Geokodierung und Erzeugung/Abfrage von Datenquellen.

4.5.7 Yahoo! Maps API

Wie Google Maps und Bing Maps bietet Yahoo! ein eigenes Karten API mit eigenen Strassenkarten, Satellitenkarten und Hybridkarten. Es wird auch ein 'Flash API'[18] und ein 'Simple API'[19] angeboten.

Die Yahoo! Maps Javascript API ist klein. Sie besteht aus Events, Koordinaten, Marker, Bilder und GeoRSS. Geometrische Formen wie ein Polygon werden nicht unterstützt. GUI Controls werden keine spezielle zur Verfügung gestellt, dafür kann allerdings die Yahoo! User Interface (YUI) Javascript-Bibliothek verwendet werden.

Yahoo! bietet API's für Geokodierung und Staumeldungen.

4.6 ALLGEMEINE BIBLIOTHEKEN

4.6.1 jQuery & jQuery Mobile

jQuery abstrahiert die Browser-Unterschiede in Javascript. Sie vereinfacht DOM Zugriffe, Event-Handling, Animationen und AJAX.

jQuery Mobile setzt auf der jQuery-Bibliothek auf und bietet spezielle Funktionalitäten für mobile Endgeräte. Gerade im Bereich Styling wurde einiges getan, um die Bedienung auf Mobilen Endgeräten so einfach und bequem wie möglich zu gestalten. Dazu gehören unter anderem grosse Buttons und Texte.

In der vorliegenden Arbeit wird jQuery mit dem Plugin jQueryRotate eingesetzt um GroundOverlays dynamisch zu drehen. jQuery Mobile kommt im Showcase zum Einsatz.

16 http://msdn.microsoft.com/en-us/library/gg191758.aspx
17 http://msdn.microsoft.com/en-us/library/ff941096%28v=VS.92%29.aspx
18 http://developer.yahoo.com/maps/flash/index.html
19 http://developer.yahoo.com/maps/simple/index.html

4.6.2 ExtJS

ExtJS ist ein Konkurrenzprodukt zu jQuery. Auch im Mobile-Bereich gibt es ein Produkt das Sencha Touch (http://www.sencha.com/products/touch/) heisst. Weitere Details zu ExtJS sind unter http://www.sencha.com/ nachzulesen.

4.6.3 YUI

YUI ist eine Javascript-Bibliothek von Yahoo! und ist ein Konkurrenzprodukt zu jQuery und ExtJS. YUI ist im interaktiven Lageplan im Einsatz, welches so übernommen wurde. Es wird eingesetzt für die Darstellung der Buttons, Events, sowie das asynchrone Laden von Daten.

4.7 OPENLAYERS MOBILE CODE SPRINT

Gleich zu Beginn unserer Bachelorarbeit wurde Seitens OpenLayers der "OpenLayers Mobile Code Sprint" durchgeführt. Damit ist OpenLayers gerüstet für Mobile Clients, sprich Smartphones wie Android, iPhone, usw.

Vor Beginn der Bachelorarbeit war Khtmlib klar führend auf Mobile Clients. Dies war einer der Gründe wieso wir uns für eine Erweiterung von Khtmlib entschieden haben. Durch den Mobile Code Sprint hat sich die Ausgangslage verändert.

Im folgenden Abschnitt werden die Features des OpenLayers Mobile Code Sprints aufgeführt.

4.7.1 Im Codesprint erstellte Features[10]

OpenLayers Mobile Beispiele: http://openlayers.org/dev/examples/mobile.html

4.7.1.1 *Mobile Compatible Controls*

OpenLayers Controls (Zoom/Karte verschieben) funktionieren ähnlich auf touch- und mausbasierten Systemen. Beispiel: http://openlayers.org/dev/examples/controls.html

4.7.1.2 *Touch Navigation Control*

Alle Touch Events (dragging, double-tapping, and tap with two fingers) werden unterstützt. Bringt grössere UI Controls.

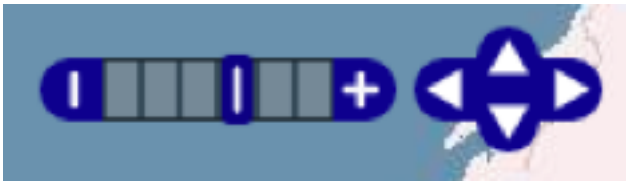


Abbildung 4.3: Bedienelement für die Karte

4.7.1.3 *Kinetic Dragging*

Nach einem Drag (auch "pan" genannt) wird die Kartenbewegung weich gestoppt.

4.7.1.4 *Pinch Zoom*

Zoomen mit zwei Fingern. (Beide Finger draufhalten und auseinanderziehen.) *Nicht möglich im Android Browser, siehe*[12].

4.7.1.5 *Anchor Permalink*

Permalinkparameter können auch über den Anchor übergeben werden.

Abbildung 4.4: Permalink per Anker

Statt als Parameter nach dem Fragezeichen (?) kommen die Parameter nach dem Anker Zeichen (#). Das ist wichtig für Javascript / AJAX Applikationen um Änderungen im Userinterface —- die keinen Seitenreload ausgelöst haben, in der Browserhistory zu verankern. Damit können die Aktionen im Browser durch den Back-Button rückgängig gemacht werden.

4.7.1.6 *GeoLocation API Support*

Über die GeoLocation API kann die Position des Users (bzw. dessen Mobiltelefon) bestimmt und/oder verfolgt werden.
GeoLocation API: `http://dev.w3.org/geo/api/spec-source.html`
Beispiel: `http://www.openlayers.org/dev/examples/geolocation.html`

4.7.1.7 *Performance*

Für die Drag-Aktionen wurden die Pixel-zu-Karte Transformationen limitiert. Damit konnte ein wichtiger performance Gewinn erreicht werden.

4.7.1.8 *Devices without Touch Control*

Minimaler Support für Geräte ohne Touch.
Unter anderem sind dies “Nokia E7”, “Windows Phone 7”.

4.8 KHTMLIB

4.8.1 Überblick

Der Code besteht im wesentlichen aus der Map Klasse. Daneben gibt es die Overlay Klassen ‘Vector’, ‘Marker’ und ‘Gpx’. Sie können als ‘Overlay’ auf die Map Klasse gelegt werden. Das gemeinsame an den Overlay Klassen sind die drei Methoden init(), render() und clear(). Bei init() wird das Overlay initialisiert und mit render() wird die Karte gerendert. Mit clear() können gerenderte Informationen wieder von der Karte entfernt werden.

Neben den Overlay Klassen gibt es weitere Hilfsklassen wie ‘Bounds’, ‘Point’ oder ‘ZoomUI’. ‘Bounds’ wird für die Angabe der Kartengrenzen verwendet. ‘Point’ ist ein Datentyp und wird für Punkte in der Karte verwendet.

Die Klassen ‘Point’ und ‘Bounds’ sind sehr einfache Klassen, weshalb nicht näher darauf eingegangen wird. Alle anderen Klassen sind jedoch weiter unten detailierter beschrieben.

4.8.2 Klasse Map

Die Klasse Map ist für die Darstellung der Karte im Browser zuständig und erwartet im Konstruktor die Übergabe eines HTML-Elements in welchem die Karte dargestellt werden soll. Die Grösse der Karte in der Webseite wird über die Grösse des im Konstruktor übergebenen Containers (beispielsweise ein div-Element) definiert.

Die Karte enthält u.a. folgende Features:

- zoomen
- draggen der Karte
- undo und redo Methoden
- Overlays

Grundsätzlich unterscheidet die Klasse zwischen zwei Arten von Positionsangaben. Einerseits existieren die Koordinaten auf der Karte in Longitude und Latitude (WGS84) andererseits existiert die Position auf der gezeigten Karte in Pixel. Der Nullpunkt

(0, 0) der Pixelkoordinaten ist die linke obere Ecke des Containers. Mit den Methoden 'XYTolatlng(x, y)' und 'latlngToXY(Point-Objekt)' können die beiden Positionsangaben ineinander umgerechnet werden.

4.8.3 Klasse Vector

Die Vektorklasse kann Flächen und Linien auf der Karte zeichnen. Je nach Browser wird entweder Canvas, SVG oder VML zur Darstellung verwendet.

Standardmässig wird SVG, für Android Canvas und für Internet Explorer VML verwendet. Der VML Support ist nicht ganz so ausgereift wie Canvas und SVG. SVG kann über CSS gestylt werden. Bei Canvas greifen diese Stylings nicht. Dafür existiert eine spezielle Funktionalität, welche im Fall von Canvas Darstellung, per Javascript alle im Browser geladenen CSS parst und die gefundenen Angaben auf das Canvas Element überträgt.

Punkte werden in einem sog. Polyline abgelegt, dieses wiederum in einem Array. Somit kann ein Vector Objekt mehrere Linien (Polylines) repräsentieren, darstellen, zeichnen. Das Zeichnen geschieht auch bei vielen Elementen flüssig. Die gesamte Funktionalität ist in dieser einen Klasse enthalten. Sie ist dementsprechend gross und unübersichtlich.

4.8.4 Klasse Gpx

Die Klasse Gpx wird verwendet um GPX-Tracks in der Karte anzuzeigen. Die Klasse erwartet bei der Initialisierung einen String oder ein DOM-Objekt mit den anzuzeigenden GPX-Daten.

Aus den übergebenen Daten wird dann die Boundingbox berechnet. Diese kann der Karte übergeben werden, so dass diese auf die Trackdaten fokussiert und zoomt. In der GPX Datei allfällig vorhandene Boundingbox Koordinaten werden ignoriert.

Zur Darstellung der Daten greift die Klasse auf die Vektorklasse zurück.

4.8.5 Klasse Marker

Die Markerklasse erlaubt es, ein beliebiges Stück HTML-Code an einer frei wählbaren Position in die Karte einzubinden. Die Klasse ist deshalb sehr mächtig und vielseitig einsetzbar. Eingefügtes wird aber beim Zoomen nicht mitskaliert, sondern nur beim ziehen der Karte mitverschoben.

Durch den Aufruf der Methode makeMoveable() kann der Marker anschliessend verschoben werden. Eine Methode um die Möglichkeit den Marker zu verschieben wieder abzustellen ist zurzeit nicht implementiert.

4.8.6 Klasse ZoomUI

Die Klasse ZoomUI bietet einen Scrollbalken über welchen die Zoomstufe der Karte per Maus angepasst werden kann. Das Aussehen ist äussert Flexibel über CSS anpassbar. Der Scrollbalken wird innerhalb der Karte dargestellt.

4.8.7 Unterstützte Karten API's

Grundsätzlich könnten Kacheln von beliebigen Servern sei es WMTS, TMS oder einem anderen Format angezogen werden. Da direkter Zugriff auf die Kacheln nicht bei allen Kartenanbietern gestattet ist, wird momentan lediglich für OpenStreetMap eine Funktion zum anziehen mitgeliefert.

5 EVALUATION OPENLAYERS MOBILE & KHTMLIB

Verglichen werden OpenLayers Version 2.10 und khtmlib Version 0.83.

5.1 KRITERIENKATALOG UND GEWICHTUNG

Siehe Tabelle 5.1 Kriterienkatalog auf Seite 38.

5.2 OPENLAYERS MOBILE

Siehe Tabelle 5.2 Bewertung OpenLayers Mobile auf Seite 39.

5.3 KHTMLIB

Siehe Tabelle 5.3 Bewertung Khtmlib auf Seite 40.

KRITERIUM	GEWICHT
MOBILE	
Touch Navigation (Zoom/Pan)	5
Spezielle UI Controls für Mobile	2
Kinetic Dragging	2
Pinch Zoom (nicht für Android)	5
GeoLocation API Support	1
Devices ohne Touch Control	3
Verwendet CSS 3D Beschleunigung	5
Performanz ohne CSS 3D	2
WEITERE FEATURES	
Permalink	3
Stufenloser Zoom	3
WMS	5
WFS	5
KATEGORIEN	
(GUI)-Controls	2
Vector Renderer	5
XmlHttpRequest Helper	1
Geometrie-Typen	5
Formate	5
Internationalisierung	3
Transformation zwischen Koordinatensystemen	5
Vektoren editieren	5
Filter	5

Tabelle 5.1: Kriterienkatalog

KRITERIUM	DETAILS	GEWICHT	PUNKTE	TOTAL
MOBILE				
Touch Navigation	Vorhanden	5	5	25
UI Controls für Mobile	Vorhanden	2	5	10
Kinetic Dragging	Vorhanden	2	5	10
Pinch Zoom (nicht für Android)	Vorhanden	5	5	25
GeoLocation API Support	Vorhanden	1	5	5
Devices ohne Touch Control	Vorhanden	3	5	15
Verwendet CSS 3D Beschleunigung	Nein	5	0	0
Performanz ohne CSS 3D	Gut	2	5	10
WEITERE FEATURES				
Permalink	Vorhanden	3	5	15
Stufenloser Zoom	Nicht vorhanden	3	0	0
WMS	Vorhanden	5	5	25
WFS	Vorhanden	5	5	25
KATEGORIEN				
(GUI)-Controls	ca. 30 Stück	2	5	10
Vector Renderer	SVG, VML, Canvas	5	5	25
XmlHttpRequest Helper	Vorhanden	1	5	5
Geometrie-Typen	10 Stück	5	5	25
Formate	29 Stück	5	5	25
Internationalisierung	Vorhanden	3	5	15
Transformation zwischen Koordinatensystemen	Vorhanden	5	5	25
Vektoren editieren	Vorhanden	5	5	25
Filter	Vorhanden	5	5	25

Tabelle 5.2: Bewertung OpenLayers Mobile

KRITERIUM	DETAILS	GEWICHT	PUNKTE	TOTAL
MOBILE				
Touch Navigation	Vorhanden	5	5	25
UI Controls für Mobile	Keine [a]	2	0	0
Kinetic Dragging	Vorhanden	2	5	10
Pinch Zoom (nicht für Android)	Vorhanden	5	5	25
GeoLocation API Support	Vorhanden	1	5	5
Devices ohne Touch Control	Vorhanden	3	5	15
Verwendet CSS 3D Beschleunigung	Vorhanden	5	5	25
Performanz ohne CSS 3D	Gut	2	5	10
WEITERE FEATURES				
Permalink	Nicht vorhanden	3	0	0
Stufenloser Zoom	Vorhanden	3	5	15
WMS	Nicht vorhanden	5	0	0
WFS	Nicht vorhanden	5	0	0
KATEGORIEN				
(GUI)-Controls	Zoombar	5	1	5
Vector Renderer	SVG, VML, Canvas	5	5	25
XmlHttpRequest Helper	Nicht vorhanden	5	0	0
Geometrie-Typen	Vektor Klasse[b]	5	3	15
Formate	GPX	5	1	5
Internationalisierung	Nicht vorhanden	3	0	0
Transformation zwischen Koordinatensystemen	Nicht vorhanden[c]	5	0	0
Vektoren editieren	Nicht vorhanden	5	0	0
Filter	Nicht vorhanden	5	0	0

Tabelle 5.3: Bewertung Khtmlib

a Khtmlib beinhaltet nur 1 Control (Zoombar). Die Zoombar funktioniert allerdings nicht auf Android. Khtmlib konzentriert sich auf die Darstellung von Karten, GUI soll anderweitig gelöst werden. Möglichkeiten zur Interaktion mit Khtmlib bestehen.

b Mit der Vektor Klasse können Linien, Flächen und Polygone dargestellt werden.

c Verwendet ausschliesslich WGS84 (GPS)

5.4 AUSWERTUNG KRITERIEN

	OPENLAYERS MOBILE	KHTMLIB
Punkte	345	180

Tabelle 5.4: Auswertung Kriterienkatalog

5.5 BEMERKUNGEN

Der Code von OpenLayers ist gut strukturiert und dokumentiert. Ein Einarbeiten in OpenLayers fällt deshalb (trotz der vielen Möglichkeiten) relativ leicht. Bei khtmlib existiert ausser dem Code und einigen Beispielen keine Dokumentation. Dank dem kleineren Codeumfang ist eine Einarbeitung trotz wenig Dokumentation noch machbar.

OpenLayers bietet für diverse Datenformate eine einfache Möglichkeit, diese in die Karte einzubinden. Für die verschiedenen Datenformate existieren verschiedene Layertypen, welche genau auf die Bedürfnisse der Daten zugeschnitten sind. Es existieren zusätzliche Features wie Popup, Highlighting usw.

Khtmlib hat keine Layer. Jedes Element kann einzeln der Karte hinzugefügt werden. Mehrere Elemente in Layern Gruppieren wird nicht unterstützt. Für Popups oder andere Overlays sind in khtmlib keine Funktionen/Klassen vorhanden. Der Code wirkt zudem etwas unorganisiert. So ist der Code mit viel auskommentierten Code-Abschnitten, sehr wenig Kommentaren sowie schlechter Einrückung sehr unübersichtlich. Es dauert entsprechend lange zu verstehen, was die Funktionen genau machen.

5.6 FAZIT

Seit dem Mobile Code Sprint ist OpenLayers auf mobilen Endgeräten ebenfalls gut gerüstet.

Khtmlib hat nebst der Ausrichtung auf Mobile Browser, sehr wenige Features. Hier soll mit dieser Arbeit angesetzt werden um einige Erweiterungen zu machen.

6 RESULTATE

6.1 ERGEBNISSE

Die gesteckten Ziele (Einarbeitung in Openlayers und Verbesserungen am Lageplan sowie Erweiterung von Khtmlib) konnten erreicht werden.

6.1.1 Interaktiver Lageplan

Die Einarbeitung in OpenLayers aufgrund des interaktiven Lageplans ist gelungen. Es konnten dadurch die verschiedenen üblichen Datenformate sowie OpenLayers kennen gelernt werden.

Der Lageplan konnte ausserdem von Fehlern befreit (Flackern von Popups behoben und Permalink geflickt) und verbessert (Zoom bei Suche, Raum markieren, Stockwerklayer hinzugefügt, Code wurde refactored) werden. Es konnten Unittest für den Code erstellt werden, sowie Sikuli-Tests um das Interface zu testen.

6.1.2 Erweiterung Khtmlib

6.1.2.1 *Begriffe*

Hier die wichtigsten Begriffe, weitere Erklärungen finden sich im Glossar.

MARKER Damit kann ein beliebiges HTML-Element an bestimmten Koordinaten auf der Karte angezeigt werden.

IMAGEMARKER Spezialform des Marker's welche ein Bild anzeigt.

OVERLAY Ein Element das auf der Karte angezeigt werden kann. (Nicht zu verwechseln mit einem Layer)

MIXIN Alternative zu klassischer Vererbung, gut geeignet für Javascript. Ermöglicht das inkludieren eines Sets von Methoden. Ermöglicht Mehrfachvererbung.

6.1.2.2 *GroundOverlay*

Abbildung 6.1: Vulkan Ätna als GroundOverlay

Dies ist ein Rasterbild das über der Karte eingeblendet wird. Wird die Karte verschoben oder gezoomt, so wird das Bild mitgezoomt oder mitverschoben.

Speziell an der Implementation ist die Fähigkeit das GroundOverlay Bild um dessen Mittelpunkt zu rotieren.

6.1.2.3 *Layer*

Ein Layer dient zum gruppieren und verschachteln von Objekten. Er kann folgende Objekte enthalten:

- weitere Layer
- LineString
- LinearRing
- Polygon
- Feature
- Marker
- ImageMarker
- GroundOverlay

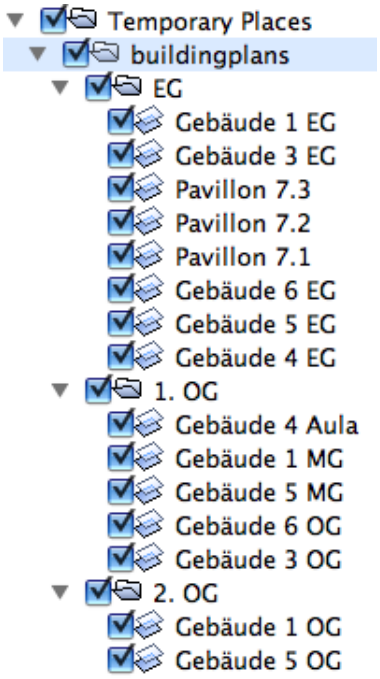


Abbildung 6.2: Ordner in Google-Earth

Grundsätzlich werden alle Objekte die das Renderable-Mixin eingebunden haben unterstützt.

Mögliche Anwendungsfälle: Abbildung der Ordnerstruktur aus KML. Einfaches einblenden, ausblenden, hinzufügen oder entfernen bestimmter Marker Kategorien.

6.1.2.4 *KML-Parser*

Der erstellte KML Parser unterstützt folgende Elemente:

- Folder
- Document
- GroundOverlay
- Point
- LineString
- LinearRing
- Polygon
- Placemark
- Icon

Der Parser kann KML lesen. Schreiben wird nicht unterstützt.

FOLDER UND DOCUMENT Die Hierarchien aus *Document* und *Folder* werden 1:1 Khtmlib Layer abgebildet. Somit bleiben die Hierarchien erhalten. Ausserdem sind alle Elemente erreichbar, egal auf welcher Hierarchiestufe sie sich befinden.

GROUNDOVERLAY Das GroundOverlay wird auf das Khtmlib GroundOverlay-Objekt abgebildet. Das GroundOverlay wird nach den Angaben aus dem KML positioniert, skaliert und falls nötig *rotiert*.

ICON Icon liefert die Bild-URL und Angaben zur Skalierung/Rotierung des GroundOverlay Bild.

PLACEMARK Das Placemark kann 0 oder mehrere KML Geometry-Elemente beinhalten. Placemarks ohne Geometry-Element werden geparst, können aber nicht auf der Karte positioniert werden. Bei mehr als 1 Element werden die restlichen verworfen.

Ist als Geometry-Element ein 'Point' vorhanden, wird ein 'Punkt' Bild an den 'Point'-Koordinaten dargestellt. Es existiert ein Hook über den das Standard-Bild verändert oder durch einen beliebigen Marker ersetzt werden kann.

Ist ein Geometry-Element vom Typ LineString oder Polygon vorhanden wird es auf das entsprechende Khtmlib Geometry Objekt abgebildet.

Sind Titel- und/oder Description-Attribut vorhanden werden sie im Feature Objekt abgelegt.

POINT, LINESTRING, LINEARRING, POLYGON Siehe vorheriger Abschnitt.

6.1.2.5 *GPX Parser*

Der KML Parser wurde auf Basis des GPX Parsers entwickelt. Die Struktur des KML Parser wurde sehr stark verbessert. Das Entfernen des View Codes aus dem KML Parser hat sich bewährt und wurde auf den GPX Parser übertragen. Der Parser ist somit selbst kein Overlay mehr, sondern liefert ein Overlay zurück. Beide Parser können über dasselbe Interface bedient werden.

6.1.2.6 *XmlHttpRequest Helper*

Der XMLHttpRequest Helper erlaubt es einfach Dateien vom Webserver zu lesen. Dies ist vorallem zum einlesen von Daten Files (KML, GPX) sehr hilfreich.

Dieses Feature verwendet jQuery.

6.1.2.7 *GeoLocation*

HTML5 bietet ein Javascript API um die aktuelle Position des Browsers als WGS84 Longitude/Latitude Koordinate zu erhalten. Dies funktioniert auf den meisten modernen Browsern, inkl. iPhone und Android.

Die GeoLocation Klasse fragt diese Koordinaten aus dem Browser ab, kapselt den Unterschied zwischen iPhone und World Wide Web Consortium (W3C) implementation. Es kann automatisch der Kartenausschnitt auf die vom Browser gelieferten Koordinaten gesetzt werden.

6.1.2.8 *Refactoring zur Vereinheitlichung des Overlay Rendering's*

Zu Beginn konnten die Vektor-Objekte, Marker und GPX-Objekte als Overlay auf die Karte gelegt werden. Jedes implementierte ein ganz Ähnliches aber trotzdem unterschiedliches Rendering. Dieses wurde vereinheitlicht und doppelter Code in das 'RenderableMixin' ausgelagert.

6.1.2.9 *Refactoring der Vektorklasse*

Die bestehende Vektorklasse hat sehr viele Funktionen an einem Ort vereinigt. Sie verwendet drei unterschiedliche Renderer und kann Linien, Flächen und Polygone zeichnen. Die Klasse hat eine Menge Code Smells[1] aufgewiesen, war schlecht Kommentiert und hatte eine fehlende Trennung zwischen Geometrie Typen (wenn man nicht aufgepasst hat wurde eine Linie als Fläche dargestellt).

Für das Handling der Geometrie Typen sind neu die Klassen LineString, LinearRing und Polygon zuständig. Die Namensgebung und Funktionalität der Typen richtet sich nach dem dem OGC 'Simple Feature Access' [2] Standard. Die Formate KML, GPX, GeoJSON verwenden ebenfalls diese Geometrie Typen. Damit wird dem KML Parser und dem für die Zukunft geplanten Support von GeoJSON, die Konvertierung erleichtert.

Das eigentliche Rendering wurde in die Klassen VectorMixin, Canvas, SVG und VML ausgelagert.

6.1.2.10 *Mobile Internet Explorer*

Als Beispiel für einen mobile Browser ohne Touch Navigation (es sind über Javascript keine Touch-Events zu Empfangen), diente uns der Mobile Internet Explorer.

1 http://en.wikipedia.org/wiki/Code_smell
2 http://www.opengeospatial.org/standards/sfa

Bis an hin war auf diesem Browser das Navigieren unmöglich. Neu ist ein simples Navigieren möglich, indem bei Klick auf die Karte, dieser Punkt an den Kartenmittelpunkt verschoben wird.

6.1.2.11 *Closure Compiler*

Der Closure Compiler erzeugt minimiertes Javascript. Es werden Zeilenumbrüche und Kommentare entfernt und Optimierungen am Quellcode vorgenommen. Zudem wird die Syntax überprüft.

Bis an hin bestand Khtmlib aus 6 Javascript Files (total 113KB). Neuerdings sind es 39 Files zu 309KB. Damit Khtmlib schneller geladen ist, wird mit dem Closure Compiler alle Javascript Files in eine einzige Datei kompiliert und verkleinert. Somit müssen weniger Daten geladen werden und es ist nur ein einziger Request nötig. Der neue Code kann mit Hilfe des Closure Compilers um einen $\frac{1}{3}$ auf 105KB verkleinert werden.

6.1.2.12 *Multitouch in Android Browser*

Auf dem Android Browser ist Multitouch (für Pinch-Zoom) nicht möglich. Sofern die Hardware Multitouch unterstützt, wird der Event vom Browser abgefangen. Der Browser verwendet Pinch-Events selbst um die Webseite zu verkleinern oder zu vergrössern.

Ein Ticket[3] das Support von Multitouch im Android Browser fordert ist auf dem Google Bug Tracker noch offen.

6.2 AUSBLICK

Khtmlib hat im Vergleich zu Openlayers noch sehr wenig Funktionalität. Es gibt noch viel Verbesserungspotenzial bei Khtmlib. Im Folgenden sind einige Ideen aufgeführt, wo bzw. in welche Richtung in Zukunft am besten weiter entwickelt werden könnte.

6.2.1 Styling

Die Bibliothek unterstützt bislang noch kein Styling der Vektoren. Eine sinnvolle Erweiterung wäre es, wenn die Bibliothek MapCSS verstehen würde und auf die Vektorzeichnungen anwenden könnte. Dazu muss ein Converter/Parser gebaut werden, der als Bindeglied zwischen MapCSS und den bereits bestehenden Anzeigeformaten Canvas/SVG/VML dient.

3 http://code.google.com/p/android/issues/detail?id=11909

6.2.2 Geometrie Typen

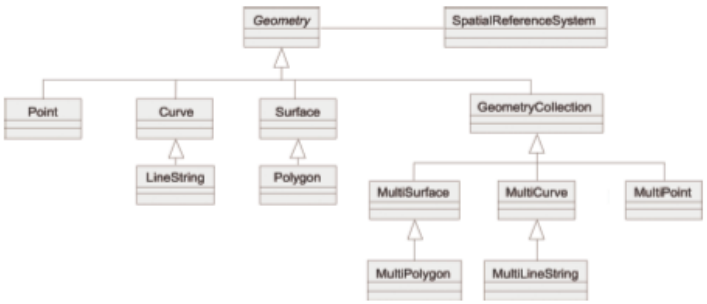


Abbildung 6.3: SQL Geometrie Typen Hierarchie[7]

Nebst den bestehenden Typen (LineString, LinearRing, Polygon) fehlen noch die GeometryCollection Typen. Der wichtigste davon ist MultiLineString. Mit MultiLineString können viele Linien performanter gerendert werden als mit LineString. Es müssen weniger Objekte erzeugt werden und auch beim SVG/Canvas rendern kann optimiert werden.

6.2.3 Boundingbox Verbesserungen

Es kann auf einen Layer gezoomt/fokussiert werden. Falls nun ein Marker in diesem Layer *genau auf den Rand* der Boundingbox zu liegen kommt, so wird dieser ausgeblendet. Das ein-/ausblenden des Markers basiert auf dessen Koordinate. Dass ein Marker nicht ein Punkt ist - sondern dessen Bild oder sonstiger Inhalt ebenfalls eine Fläche bedeckt, sprich eine Boundingbox haben sollte, wird nicht berücksichtigt.

Das gedrehte GroundOverlay liefert die Boundingbox des *ungedrehten* GroundOverlays zurück. Dies sollte in der Methode 'GroundOverlay.bounds()' verbessert werden. Es müsste das grosse schwarze Rechteck (Bounded Box) aus Abbildung 6.4 (S. 50) berechnet werden.

4 Quelle: http://stackoverflow.com/questions/622140/calculate-bounding-box-coordinates-from-a-rotated-rectangle-picture-inside

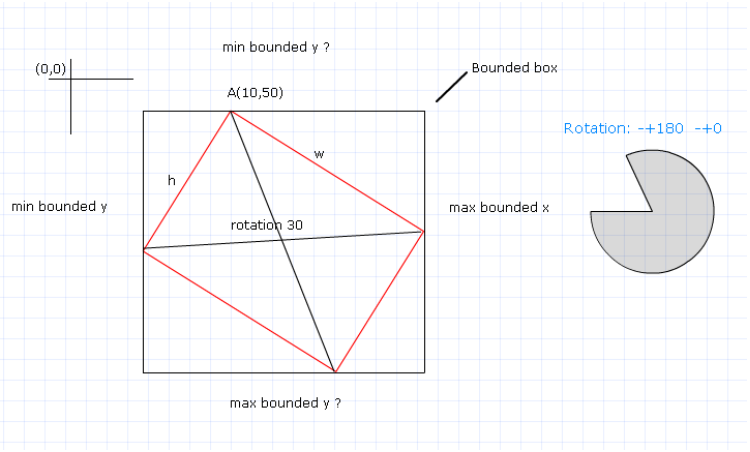


Abbildung 6.4: Boundingbox[4]

6.2.4 Liste offener Tasks

- WMS: Clientseitiges zoomen — Beim Panning wird das Bild Clientseitig mitverschoben. Analog sollte beim Zoomen, das Bild Clientseitig gezoomt werden. Dies als Zwischenschritt bevor das neu gerenderte Bild vom Server eintrifft.
- Marker: Popups — Die Bibliothek bietet keine direkte Möglichkeit Popups zu erstellen welche angezeigt werden, sobald über ein POI gefahren wird. Grundsätzlich kann eine solche Funktionalität bereits heute schon mit der Marker-Klasse erreicht werden. Dazu ist aber noch viel Handarbeit des Benutzers nötig. Dies könnte vereinfacht werden.
- KML
  - Fehler in IE flicken — Der Parser hat noch Fehler im Internet Explorer. Sehr wahrscheinlich handelt es sich um Probleme beim Zugriff auf DOM Elemente.
  - Placemark: mehrere Geometrie-Elemente — Laut Standard kann ein Placemark mehrere Geometrie-Elemente enthalten. Aktuell würde nur eines Verwendet wenn mehrere vorhanden sind.
  - Weitere Teile des KML Standards unterstützen.
- GeoJSON support — Bislang werden nur GPX-Tracks sowie KML-Dateien für den Import und die Anzeige in der Karte unterstützt. Die Erweiterung des Supports auf GeoJSON-Daten wäre naheliegend und auch mit relativ wenig Aufwand hin zu bekommen.
- GPX:
  - Parser auf neue Geometrie Objekte umstellen. Nicht mehr khtml.maplib.overlay.Vector verwenden.
  - Weitere Elemente parsen — Aktuell werden Track Segmente gelesen. Hier wäre ein Ausbau auf weitere GPX Elemente möglich.
- Zoombar: Android unterstützen — Die Zoombar verarbeitet aktuell nur mouse* Events. Für Touchscreens müssen auch die touch* Events verarbeitet werden.
- CSS3D Transformationen für Android — Analog zu den CSS3D Transformationen für mobile Safari, für Android Webkit implementieren. Falls bereits machbar und sinnvoll.

- Vektor Geometrie Klassen (LineString, LinearRing, Polygon) — API ausbauen. Aktuell sind nur die notwendigsten Methoden zum Punkte hinzufügen vorhanden. Methoden zum lesen und bearbeiten der Punkte wären wünschenswert.
- Vektor Rendering:
  - Freeze/Overload-Schutz implementieren. Die Map Klasse implementiert Beispielsweise etwas ähnliches, wenn der Browser beim rendern zu fest beansprucht wird, werden gewisse Renderingvorgäng ausgelassen. Ein Freeze-Schutz soll während dem Vektor rendern sicherstellen, dass die Karte immernoch bedienbar ist (zoomen, panning).
  - Vektor Rendering: gute kommentierte Beispiele in das Tutorial übernehmen. Wie man bei den manuellen Testcases sieht, ist sehr viel möglich. Dies sollte den Anwendern bewusst gemacht werden.
- Tilesource: möglichkeit die Transparenz des Tile-Overlays zu konfigurieren. Die Option 'opacity' aus dem Tilesource-Objekt sollte berücksichtigt werden. Per CSS Bildtransparenz[5] sollen die Kacheln dann transparent geschaltet werden.

6.3 PERSÖNLICHE BERICHTE

Die Persönlichen Berichte befinden sich im Anhang A (Seite 121).

6.4 DANK

Dankbar blicken wir auf die Zusammenarbeit mit Prof. Stefan F. Keller und Bernhard Zwischenbrugger zurück. Herr Keller hat unsere Arbeit betreut und ist uns beratend zur Seite gestanden. Herr Zwischenbrugger (Maintainer von Khtmlib) hat sich Zeit genommen unsere Entwicklungen zu Begutachen und uns Feedback zu geben.

Vielen dank für die Unterstützung.

5 http://www.w3schools.com/Css/css_image_transparency.asp

Teil III

SOFTWARE-PROJEKTDOKUMENTATION VORPROJEKT LAGEPLAN

Als Vorprojekt wurde gefordert, den interaktiven Lageplan der HSR zu verbessern und sich gleichzeitig mit OpenLayers vertraut zu machen. Im vorliegenden Kapitel ist die Dokumentation um interaktiven Lageplan zu finden.

6.5 ANFORDERUNGSSPEZIFIKATION

6.5.1 Ausgangslage

Es existierte bereits eine Webseite mit einem Lageplan. Dieser Lageplan konnte folgendes:

- Suche nach Raum (Details wurden angezeigt und Gebäude blau hervorgehoben)
- Suche nach Zugverbindungen via SBB
- Beschriftung von Gebäuden und POIs (flackerten bei Mouseover)

Folgende Punkte sollen verbessert werden:

- Das Flackern von Gebäude & POI Beschriftungen soll verhindert werden.
- Die Suche nach einem Raum soll auch mit der Enter-Taste abgesetzt werden können.
- Neben dem Gebäude soll auch der gesuchte Raum in der Karte hervorgehoben werden.
- Die Karte soll auf den gefundenen Raum bzw. auf das Gebäude fokussieren und Zoomen.
- Der Code soll auf die neue OpenLayers Version angepasst und refactored werden.
- Die Karte soll mit Stockwerklayern versehen werden.
- Der Permalink soll richtig funktionieren (inkl. Speicherung des Suchresultats).
- Karte soll den Designrichtlinien der HSR angepasst werden. können.
- Die Applikation muss in den Browsern IE8, Firefox 3.6, Safari 4 und evt. Chrome lauffähig sein.
- Permalink

6.6 ANALYSE

6.6.1 Refactoring

Folgende Punkte wurden nach einer Analyse des bestehenden Codes als verbesserungswürdig eingestuft:

- Ordnerstruktur: alle Dateien (Javascript, CSS, Bilder) sind in einem einzigen Ordner.
- zu alte OpenLayers Version
- Javascript Code in 'map.js':
  1. Keine Kommentare vorhanden
  2. Include des Javascript-Files (map.js) löst sofort die Ausführung des Codes aus.
  3. Keine Konfiguration möglich
  4. Globaler Namensraum wird mit Methoden und Variablen übersät. Dies birgt Konfliktpotenzial mit Javascript Code der von anderen Quellen stammt, bzw. falls die Karte auf einer bestehenden Seite integriert werden soll.
  5. Durcheinander: zusammengehörende Funktionen sind an beliebigen Orten platziert
  6. Dieselben Dateinamen, etc. werden an mehreren Stellen definiert. Verletzung des Don't Repeat Yourself (DRY) Prinzips.

Zur Behebung dieser Defizite wurden folgende Massnahmen durchgeführt:

- Pro Dateityp ein Unterordner erstellt
- aktuellste OpenLayers Version integriert
- Refactoring: map.js
  1. Kommentare hinzugefügt
  2. Funktionen in logische Einheiten gruppiert. Mithilfe des Object-Literal Patterns[6]. (behebt 5. Durcheinander und 4. Verschmutzung des globalen Namensraums)
  3. Konfigurierbarkeit: wird durch verschiedene Massnahmen erreicht.

6 http://www.brainonfire.net/blog/javascript-object-literal-namespace/

a) Zu konfigurierende Daten (Startkoordinaten, Dateinamen, etc.) in zentrales Settings-Objekt verschoben.

b) Das Javascript-File führt sich nicht mehr selbst aus. Die init() Methode muss von aussen - nachdem z.B. Konfigurationen vorgenommen worden sind - aufgerufen werden.

(behebt 7. Problem mit doppelten Dateinamen und 3. fehlende Konfigurationsmöglichkeiten)

6.7 DESIGN

Der Interaktive Lageplan greift mit Hilfe von OpenLayers auf die Kartendaten von Openstreetmap zu. Neben den OSM-Daten wird der Plan mit lokal gespeicherten Geodaten angereichert, welche mit YUI asynchron geladen und mit OpenLayers aufbereitet und angezeigt werden.

Bei den Geodaten handelt es sich um JSON, OSM und KML Files. In den JSON Files sind Gebäude-, Raum- und POI-Informationen abgelegt. Bei den Rauminformationen handelt es sich im wesentlichen um eine Zuordnung von Zimmern zu den Gebäuden. Ebenfalls in rooms.json enthalten ist das Zentrum von jedem Gebäude. Die OSM-Dateien enthalten Information über die Position der Türen von einzelnen Zimmern. Die Files rooms.json und HSR_OSM_Daten.osm werden für die Raumsuche benötigt (Hintergründe und Details im Kapitel Raumsuche).

Die KML Files enthalten Pfade zu Bildern von Bauplänen. Diese werden in den Stockwerklayern Erdgeschoss, 1. Stock und 2. Stock verwendet.

Die Verwendung der verschiedenen Datenformate ist historisch bedingt. Die JSON-Dateien existierten bereits. Die Position der Türen von Räumen konnte bei Openstreetmap nur als OSM-File exportiert werden. Die georeferenzierten Bilder der Baupläne erhielten wir mit KML-Dateien. Da OpenLayers mit allen Datenformaten zurecht kommt, verzichteten wir aus Zeitgründen, die Daten in ein gleiches Format zu bringen.

6.7.1 Raumsuche

In den Funktionen für die Raumsuche enthalten ist auch die Logik der Raumnummerierung welche an der HSR üblich ist. Durch diese hart codierte Logik wird es möglich, auch bei nicht erfassten Räumen das dazu gehörige Gebäude zu finden. Damit ein Fokussieren des Gebäudes möglich wird (wenn keine Positionsangaben zur Raumtüre hinterlegt sind), müssen in der Datei rooms.json zu jedem Gebäude auch die Koordinaten hinterlegt sein. In der Datei rooms.json sind also sowohl Gebäudekoordinaten wie auch zusätzliche Rauminformationen gespeichert. Ebenfalls gespeichert ist, welcher Raum zu welchem Gebäude gehört.

Die Datei HSR_OSM_Daten.osm enthält alle Koordinaten aller Türen. Wird eine zum gesuchten Raum zugehörige Türe gefunden, so wird diese Türe (ist als Punkt hinterlegt) auf der Karte rot markiert und ins Zentrum der Karte gerückt.

Eine suche funktioniert theoretisch ohne die Datei HSR_OSM_Daten.osm, kann dann aber die Türe zum Raum nicht markieren. Hingegen wird von der Suche zwingend die Datei rooms.json benötigt.

6.7.2 Layerstruktur Baupläne

Weil OpenLayers GroundOverlays in KML nicht unterstützt, konnten die Baupläne nicht in einen KML-Layer dargestellt werden. Jeder Bauplan musste in einen eigenen Image-Layer gepackt werden. Zum aus/einblenden von allen Bauplänen des Erdgeschosses müssen beispielsweise mehrere Layer ausgeblendet werden. Der Funktion zum ein/ausblenden von Layers wird ein Array mit Layers mitgegeben.

Die Baupläne befinden sich bei einem Z-Index von 500 an aufwärts.

6.7.3 Debugging

Für Debuggingzwecke kann in der URL $\&debug = true$ angehängt werden. Dies Bewirkt, dass alle erfassten Türen der Räume angezeigt werden. Zusätzlich erscheinen bei 'Angezeigte Informationen' der Knopf Zimmer womit diese ein/ausgeblendet werden können.

6.8 TESTING

6.8.1 Wahl der Testing-Tools

Für das Unit-Testing des JavaScript-Codes wurde Jasmine eingesetzt. Jasmine ist das Nachfolgerprodukt von JSUnit welches von Herr Keller für das Unit-Testing vorgeschlagen wurde. Für die visuelle Überprüfung der Kartenapplikation ist Jasmine jedoch ungeeignet.

Jasmine wurde gewählt, weil das Tool keine Forderungen an die Umgebung stellt (es ist kein DOM nötig, kann also auch ausserhalb eines Webbrowser verwendet werden). Zudem hat uns die BDD[7] Test-Methode sehr angesprochen. Einerseits können einfach Anforderungen in Testcode übertragen werden. Andererseits wird der Testcode sehr selbstsprechend und daher einfach zu lesen.

Für die Visuelle Überprüfung wird (wenn nötig) Sikuli eingesetzt.

Selenium haben wir ebenfalls in betracht gezogen. Jedoch haben wir mit Sikuli und Jasmine bereits alle Bereich abgedeckt.

6.8.1.1 *Selenium*

Mit Selenium können fertige Webseiten getestet werden. Es erlaubt einem *alle* Interaktionen die ein Benutzer durchführen kann zu simulieren.

Repetitive Tasks wie auf einer Webseite einloggen, ein Formular ausfüllen usw. können damit automatisiert werden.

Zur Überprüfung ob die Aktionen wie gewollt durchgeführt wurden können diverse Assertions verwendet werden. Z.B. prüfen auf welcher Seite (URL) man sich befindet, existiert ein bestimmtes HTML Element auf der Seite, ist die Checkbox gesetzt, kommt ein bestimmter Text in der aktuellen Webseite vor, usw.

Es existiert ein Firefox-Plugin, welches es erlaubt, sämtliche Aktionen im Browser aufzunehmen und danach wieder abzuspielen. Text und Elemente können im Browser markiert und per Rechtsklick einer Überprüfung in einem Testcase unterzogen werden.

Eine echte visuelle Überprüfung des Aussehens der Karte ist mit Selenium aber nicht möglich, da keine Bilder überprüft werden können.

7 http://dannorth.net/introducing-bdd/

6.8.1.2 *Sikuli*

Sikuli ist ein grafisch orientiertes Testing-Tool. Es arbeitet mit Printscreens. Die Printscreens - beliebig grosse Ausschnitte des Bildschirms - lassen sich einfach aufzeichnen. Es werden Aktionen auf die Printscreens definiert (wie beispielsweise klicken). Überprüfungen des UI funktionieren auch aufgrund von Printscreens. Das Tool vergleicht das Aussehen mit dem erwarteten Printscreen.

Das Problem bei Sikuli ist die Ausgangslage, welche nicht definiert werden kann. Es ist nicht möglich im Test selbst zu definieren, dass der Browser geöffnet und die Kartenseite geladen sein muss. Eine weitere Schwierigkeit ist, dass die Tests fehlschlagen, wenn gleichartig aussehende Elemente ausserhalb der zu testenden Kartenapplikation vorhanden sind. Sikuli nimmt bei mehrfachem vorkommen des gespeicherten Bildschirmausschnittes einfach den ersten Ausschnitt, welcher gefunden wird. In den meisten Fällen ist dieses Verhalten nicht weiter schlimm und kann durch grössere Printscreens oder durch schliessen von anderen offenen Applikationen mehrheitlich behoben werden.

6.8.2 Testcases - Interaktiver Lageplan

- Raum suchen und in Karte markieren (Gebäude und Raumeingang)
- Layer ein/ausblenden
- Beschriftung von gefundenem Raum bei Mouseover anzeigen und ausblenden bei Mouseout
- Beschriftungen von POIs bei Mousover ein- und bei Mouseout ausblenden
- Anreise mit SBB raussuchen
- Permalink

6.8.2.1 *Testumgebung*

Zu Testen in folgenden Browsern:

- Firefox
- IE8 (kann nur unter Windows getestet werden)
- Safari

6.8.2.2 *Art der Tests*

Die Tests sind darauf ausgelegt nur das Zusammenspiel der verschiedenen Komponenten zu überprüfen. Die einzelnen Komponenten werden nicht unabhängig voneinander getestet. Dazu hätte der bestehende Code noch viel besser in zu testende Units unterteil werden müssen. Die Codebasis für den interaktiven Lageplan haben wir übernommen. Durch ein Refactoring konnte die Kopplung des Codes bereits reduziert werden. Für das Testen einzelner Komponenten wäre eine weitere Entkopplung der Bestandteile nötig gewesen weshalb nun vorallem Integrationstests durchgeführt werden.

6.8.2.3 *Probleme*

Da der Interaktive Lageplan die Layer asynchron lädt, kam es Anfangs vor, dass Tests fehlschlugen, weil noch nicht alle Layers geladen waren. Dieses Problem lösten wir zuerst mit Timeouts. Später sorgten wir dafür, dass die Karte nur einmal initialisiert wird (nicht bei jedem Test wieder neu).

Die Testcases sind so organisiert, dass zuerst die Dinge getestet werden, welche noch nicht die geladenen Layer voraussetzen. Dadurch erhält die Kartenapplikation genügend Zeit alle Layers zu laden.

Visuelle Effekte konnten mit Jasmine nicht oder nur teilweise überprüft werden.

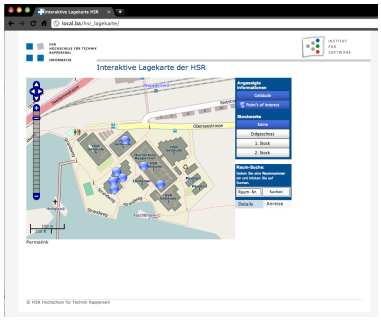
6.8.2.4 *Testprotokoll - manuelle Tests in den geforderten Browsern*

	Test Cases	IE8 WIN7	FF3.6 WIN7	FF3.6 OSX	Safari4 OSX	Chrome10 OSX
1)	Raum 1.262 suchen	OK	OK	OK	OK	OK
2)	Nicht existierenden Raum suchen	OK	OK	OK	OK	OK
3)	Link für Raum 6.108 erzeugen (Permalink)	OK	OK	OK	OK	OK
4)	Beschriftung Suchresultat-Icon anzeigen	OK	OK	OK	OK	OK
5)	Beschriftung POI anzeigen	OK	OK	OK	OK	OK
6)	Anreise mit SBB raussuchen	OK	OK	OK	OK	OK
7)	Karte von Stockwerk 2. anzeigen	OK	OK	OK	OK	OK

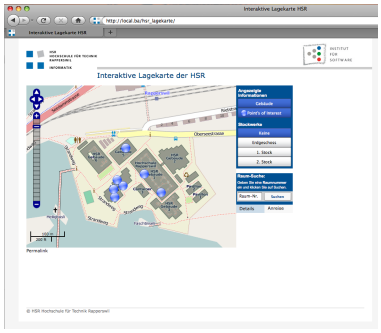
Abbildung 6.5: Browsertests Lageplan: Resultat

	Test Cases		
1)	**Raum 1.262 suchen**	1.1	Seite öffnen
		1.2	Klick in Suchfeld und "1.262" eintippen
		1.3.	Suchbutton klicken
		1.3.	Enter drücken
		1.4	Karte zoomt auf Raum 1.262 das Gebäude wird gehilightet an Position von Raum 1.262 erscheint ein roter Punkt
2)	**Nicht existierenden Raum suchen**	2.1	wie bei 1) - diesmal mit Raumnummer 9.999
		2.2	Es wird ein Alert mit Hinweis auf ungültige Raumnummer angezeigt
3)	**Link für Raum 6.108 erzeugen (Permalink)**	3.1	wie bei 1) - diesmal mit Raumnummer 6.108 statt 1.262
		3.2	Auf den Link "Permalink" klicken
		3.3	Karte öffnet sich nun unter Permalink-URL mit denselben Einstellungen (Angezeigte Layer, Suchabfrage, Position, Zoomlevel) wie vorher.
4)	**Beschriftung Suchresultat-Icon anzeigen**	4.1	Suche absetzen wie bei bei 1)
		4.2	Mit Maus über gefundenen Raum fahren: die Raumnummer wird nun eingeblendet
		4.3	Verlässt die Maus das Raum Icon wird die Nummer ausgeblendet
5)	**Beschriftung POI anzeigen**	5.1	Mit Maus über POI fahren: die Beschreibung wird nun eingeblendet
		5.2	Verlässt die Maus das Raum Icon wird die Beschreibung ausgeblendet
6)	**Anreise mit SBB raussuchen**	6.1	Seite öffnen
		6.2	Auf "Anreise" klicken, nun wird das SBB Suchformular angzeigt
		6.3	Suche eingeben und absetzen
		6.4	Suchresultat wird in neuem Fenster geöffnet
7)	**Karte von Stockwerk 2. anzeigen**	7.1	Karte öffnen, Stockwerk Button "Keine" ist aktiv
		7.2	Klick auf Button "1. Stock": Karte für 1. Stock wird angezeigt
		7.3	Klick auf Button "1. Stock": Karte für 1. Stock wird ausgeblendet, Karte für 2. Stock wird angezeigt

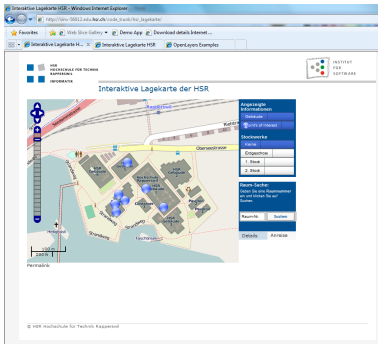
Abbildung 6.6: Browsertests Lageplan: Beschreibung



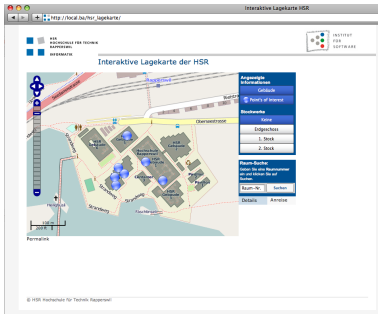
(a) Chrome 10 (OSX)



(b) Firefox 3.6 (OSX)



(c) IE 8 (Win7)



(d) Safari 4 (OSX)

Abbildung 6.7: Lageplan in verschiedenen Browsern

6.8.2.5 *Testprotokolle Sikuli Tests*

Im Folgenden ist eine tabellarische Auflistung mit den verschiedenen Testcases zu finden, welche mit Sikuli getestet wurden, sowie das Resultat für jeden Browser:

Testcases	Firefox	Safari	IE8	Chrome
Suche	ok	ok	nicht getestet	ok
Layersichtbarkeit	ok	ok	nicht getestet	ok

Beim Testcase Suche wird der Raum 6.108 gesucht und anschliessend überprüft, ob korrekt gezoomt und fokusiert wurde. Der Testcase überprüft auch, ob das Label des gehighlighteden Raumes korrekt angezeigt wird.

Beim Testcase Layersichtbarkeit werden die Stockwerklayer ein/ausgeblendet per Klick auf die dafür vorgesehenen Buttons. Es wird überprüft ob die Layer korrekt ein/ausgeblendet wurden.

Die Sikuli-Tests wurden für die Durchführung auf OS X (Apple-Betriebssystem) konzipiert. Da IE8 nicht für Appel-Betriebssysteme verfügbar ist, wurde auf den Test von IE8 mit der Sikuli-Testsuite verzichtet. Stattdessen wurde manuell überprüft ob alles einwandfrei Funktioniert. Die Manuellen Tests wahren ebenfalls erfolgreich.

6.9 FAZIT

Rückblickend auf die Arbeiten am interaktiven Lageplan kann man festhalten, dass die grössten Probleme im Zusammenhang mit der Layerstruktur von OpenLayers aufgetreten ist. So ist zum Teil das verhalten nicht intuitiv und leider auch nicht genügend dokumentiert.

So lässt sich die Layer-Reihenfolge beispielsweise nur dann mit z-index verändern, wenn das SelectFeature control nicht über mehrere Layers angewandt wird. OpenLayers packt alle Layers auf welchen das SelectFeature control aktiviert wurde in ein SVG-Bild und nicht in unterschiedliche div-Layers. Mit der Zeit, welche in der Arbeit zur Verfügung stand, konnte kein Weg gefunden werden, die Reihenfolge der g-Tags im SVG-Bild nach der Initialisierung der Karte noch zu ändern.

OpenLayers unterstützt sehr viele Datenformate. Es ist einfach Daten aus Files nachzuladen. Bezüglich der Unterstützung

von KML sind wir aber bei OpenLayers an Grenzen gestossen. So werden beispielsweise groundOverlays nicht durch die OpenLayersbibliothek unterstützt. Um die Baupläne welche im KML-Format gespeichert waren einbinden zu können, musste mit dem XML-Parser von OpenLayers das KML-File analysiert werden und die einzelnen Baupläne in Image-Layers eingebunden werden.

Zusammenfassend kann man sagen: OpenLayers ist eine mächtige Bibliothek mit vielen unterstützten Datentypen und Möglichkeiten. Die Layerstruktur könnte besser dokumentiert sein und bei KML werden nicht alle Tags unterstützt. Alles in allem hinterlässt OpenLayers einen ausgereiften und stabilen Eindruck. Entwickeln mit OpenLayers ist trotz der relativ komplexen Thematik verhältnismässig einfach.

Teil IV

SOFTWARE-PROJEKTDOKUMENTATION HAUPTPROJEKT KHTMLIB

7 VISION

Siehe Abschnitt 3.1 auf Seite 17.

8 ANFORDERUNGSSPEZIFIKATION

8.1 ANFORDERUNG AN DIE ARBEIT

Anforderungen an die Arbeit im Allgemeinen (zu wählende Sprache, Termine, Projektadministration, etc.) sind durch die Aufgabenstellung (siehe Kapitel 1) gegeben.

8.2 FUNKTIONALE ANFORDERUNGEN

- GroundOverlay darstellen: (Raster-)Bild auf der Karte darstellen. Bild kann rotiert werden.
- KML darstellen: Daten aus KML File einlesen und auf der Karte darstellen.
- WMS darstellen: Ein WMS Overlay auf der Karte anzeigen können.

8.2.1 Zu unterstützende KML Features in Khtmlib

- Erste Priorität
  Zuerst sollen grundsätzliche Geometrie Features dargestellt werden können.
  - GroundOverlay
  - Folder
  - Document
  - Point
  - LineString
  - Polygon
  - Placemark
  - Icon
- Zweite Priorität
  Wenn Features dargestellt werden können, kann in einem zweiten Schritt deren Darstellung verändert werden.
  - StyleSelector

  - BallonStyle
  - ListStyle
  - LineStyle
  - PolyStyle
  - IconStyle
  - LabelStyle

- Dritte Priorität
  Alle Zeitangaben, 3D Positionen sowie Tracks und Playlists haben die niedrigste Priorität.
  - gx:Track
  - gx:Tour
  - gx:TimeSpan
  - gx:Playlist
  - Camera
  - LookAt
  - etc.

Diese Auswahl dient als Grundlage zur Priorisierung der KML-Features. Welche Features davon umgesetzt werden, muss während dem Projektverlauf entschieden werden.
KML soll nur gelesen und dargestellt werden.

8.3 NICHT-FUNKTIONALE ANFORDERUNGEN

8.3.1 Browser

Folgende Browser sollen unterstützt werden. Das bedeutet, dass die umgesetzten Erweiterungen auf diesen Browsern lauffähig sein sollen.

- Firefox >= Version 3.5
- Safari >= Version 5
- Android Webkit
  - Auf den uns zugänglichen Testgeräten (HTC Desire Z, Google Nexus One)
  - HTC Desire Z: Webkit 533.1 (Android 2.2.1)
  - Google Nexus One: Webkit 533.1 (Android 2.3.4)

- Internet Explorer >= 8
  - Bisherige Funktionalität soll weiterhin funktionieren.
  - Kompatibilität der Neuerungen wünschenswert, ist jedoch optional.

8.3.2 Installation

Die Installation soll so einfach wie möglich sein. Dazu wird:

- Die Installation in der Anleitung beschrieben
- CSS File mit sinnvollen Defaults erstellt
- Alle für die Installation nötigen Dateien per Buildscript in separaten Ordner kopieren
- Gesamter Code wird in 1 Javascript File kopiert

8.3.3 Dateigrösse

Durch den Einsatz eines "Minifiers" oder Compilers soll die Javascript-Dateigrösse reduziert werden.

8.4 USE-CASES

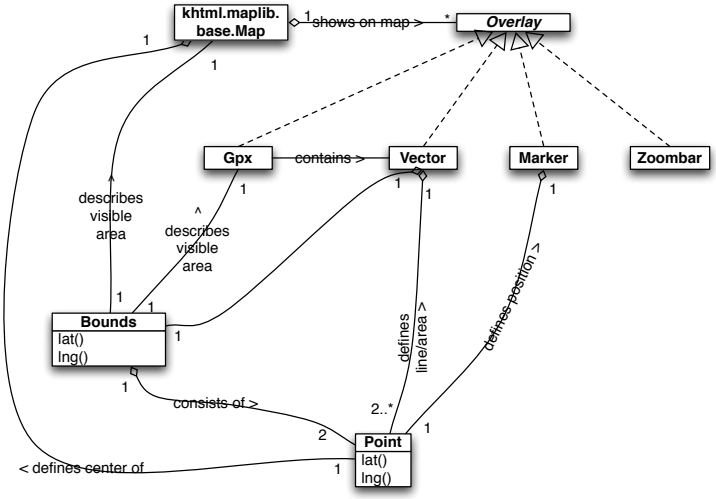
Es sind keine Use-Cases vorhanden.

9 ANALYSE

9.1 DOMAIN MODELL

9.1.1 Bisher

Abbildung 9.1: Domain Model (bisher)



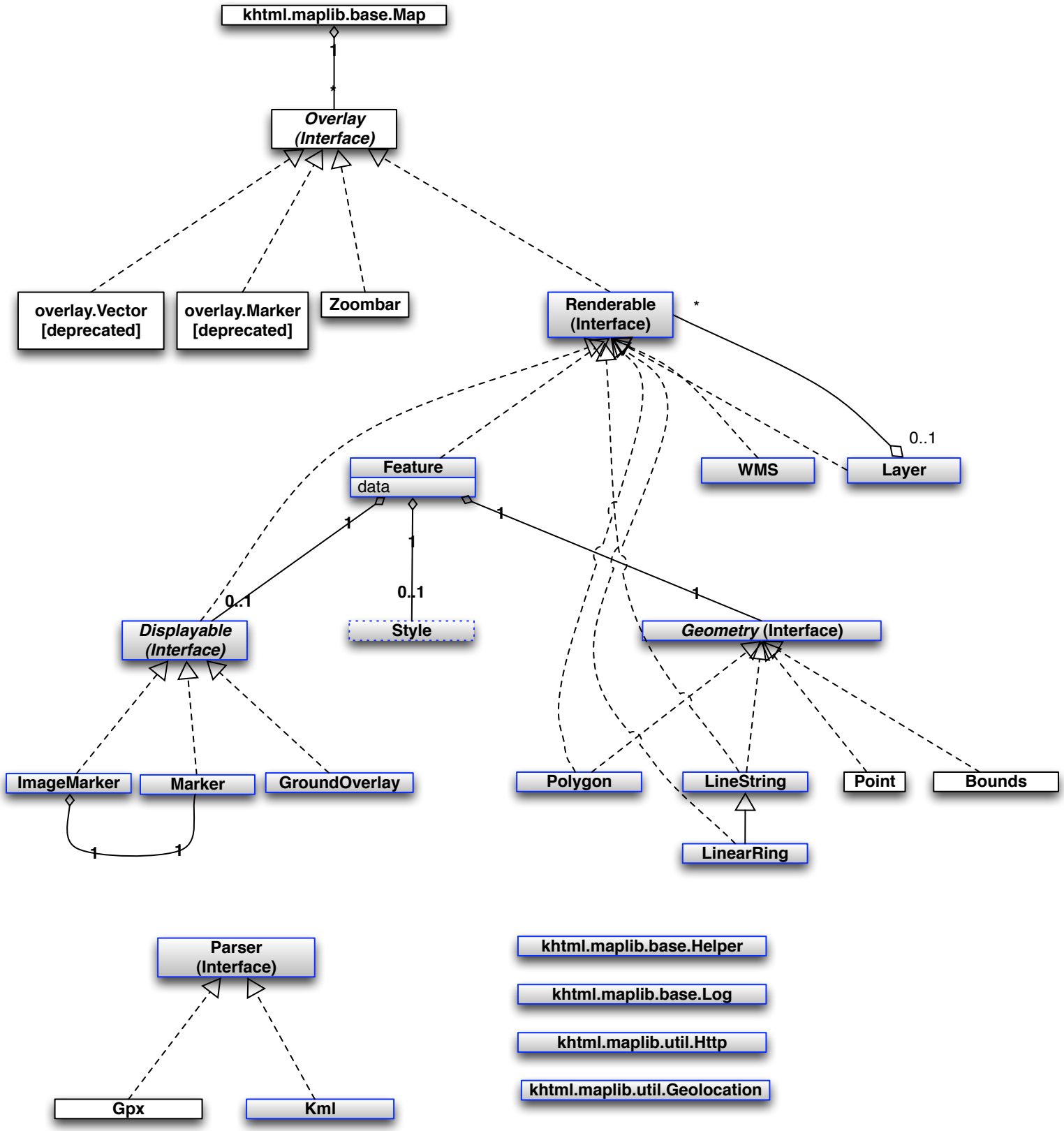
Dieses Model bildet das Domain Model ab welches wir zu Beginn der Arbeit bei Khtmlib angetroffen haben.

Javascript ist eine dynamisch typisierte Sprache. Dementsprechend ist das Interface "Overlay" als "duck type" [1] implementiert. Alle Objekte welche als Overlay behandelt werden implementieren die nötigen Methoden.

1 http://en.wikipedia.org/wiki/Duck_typing

9.1.2 Neu

Dieses Diagramm beschreibt das endgültige Domain Model mit allen Klassen und den wichtigsten Beziehungen.

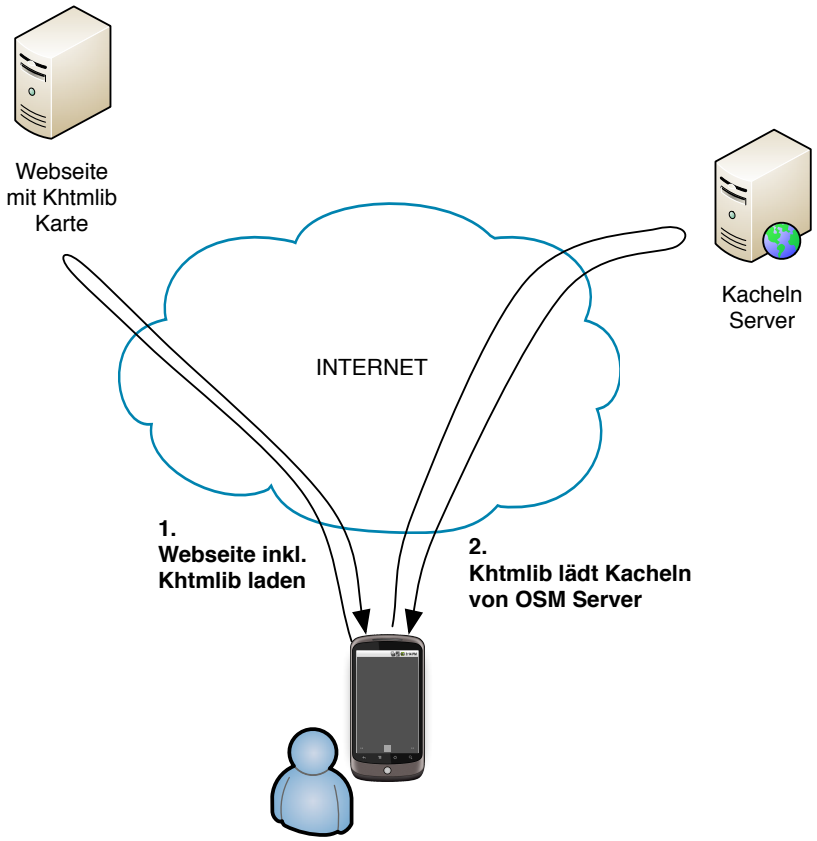

khtml.maplib.base.Map
1
*
Overlay (Interface)
overlay.Vector [deprecated]
overlay.Marker [deprecated]
Zoombar
Renderable (Interface)
*
0..1
Feature
data
WMS
Layer
1
1
1
0..1
0..1
1
Displayable (Interface)
Style
Geometry (Interface)
ImageMarker
Marker
GroundOverlay
Polygon
LineString
Point
Bounds
1
1
LinearRing
Parser (Interface)
Gpx
Kml
khtml.maplib.base.Helper
khtml.maplib.base.Log
khtml.maplib.util.Http
khtml.maplib.util.Geolocation
Legende:
Bestehend
NEU - bereits implementiert
NEU - nicht implementiert

10 DESIGN

10.1 PHYSISCHE ARCHITEKTUR

Der Benutzer ruft über seinen Browser eine Webseite auf in der eine Khtmlib Karte eingebettet ist. Khtmlib wiederum lädt die Kartenbilder vom Kachel Server (z.B. von OpenStreetMap).

Abbildung 10.1: Physische Architektur

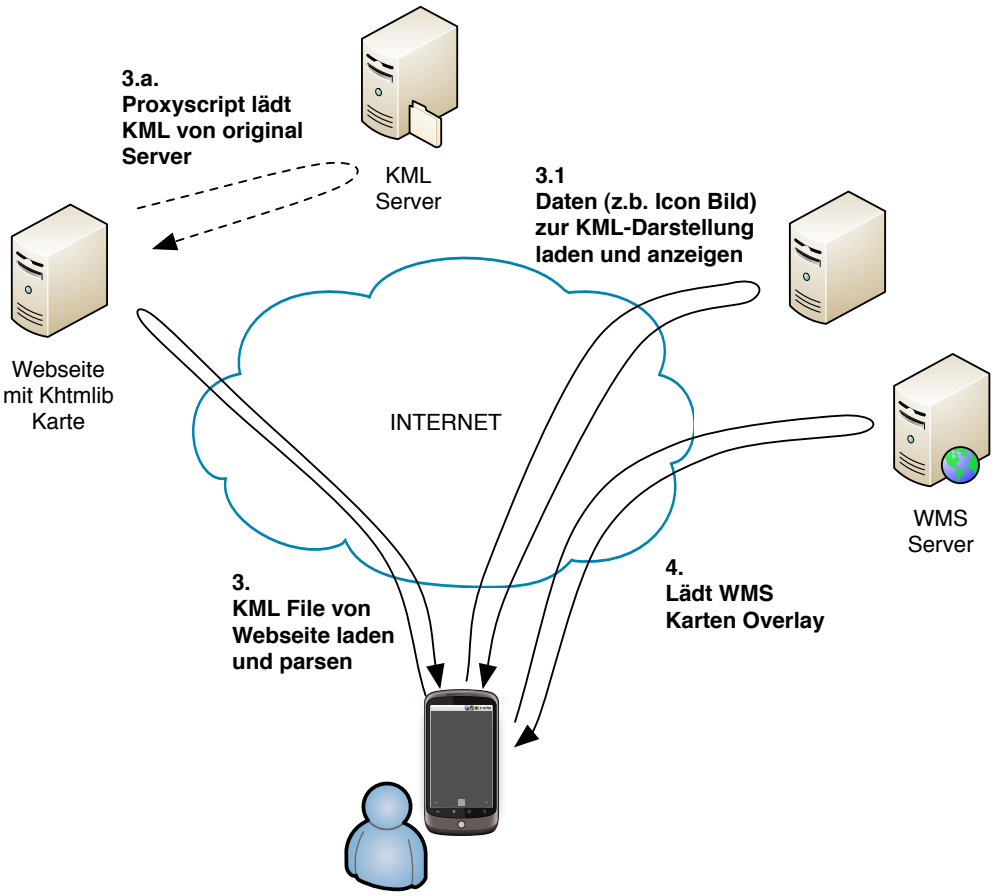


Die Verwendung von KML und WMS wird im zweiten Diagramm aufgezeigt. Das KML File muss immer vom selben Server wie die Webseite gelesen werden. Das ist eine Einschränkung des XmlHttpRequests aufgrund der Same-Origin Policy[1]

1 https://developer.mozilla.org/en/Same_origin_policy_for_JavaScript

im Browser. Soll ein KML von einem anderen Host geladen werden, so bietet sich das Cross Domain Pattern[2] in Form eines Proxyscripts auf dem Kartenserver an (3.a) (Script ist nicht bestandteil von Khtmlib).

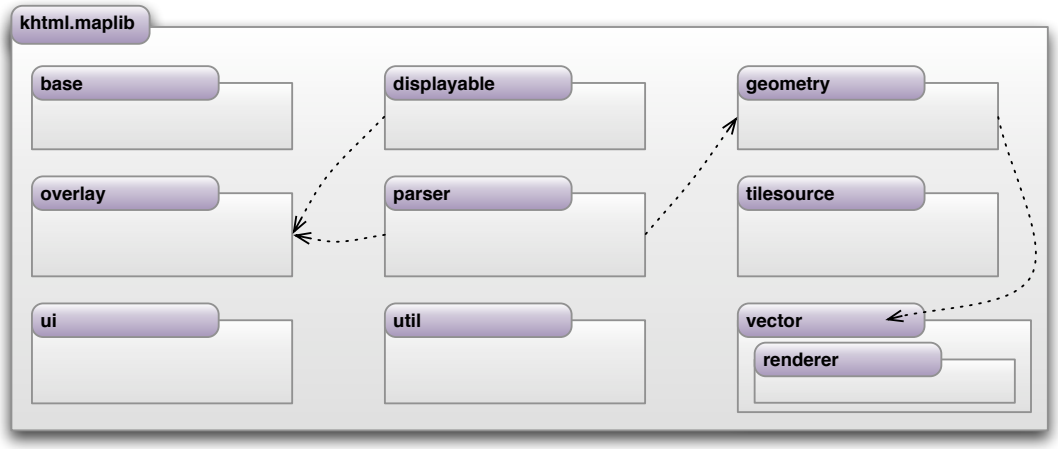
Abbildung 10.2: Physische Architektur - KML und WMS



2 http://ajaxpatterns.org/Cross-Domain_Proxy

10.2 LOGISCHE ARCHITEKTUR

Abbildung 10.3: Package Diagramm



Hinweis: Alle Packages sind von "base" abhängig. Für eine bessere Übersicht wird diese Abhängigkeit nicht dargestellt.

Die Packages sind in Tabelle 10.1 (S. 82) beschrieben.

PACKAGE	BESCHREIBUNG
base	Grundlegende (Karten-)Funktionalität. Diese Klassen sollten immer inkludiert werden.
displayable	Objekte dieser Klassen werden sichtbar auf der Karte dargestellt.
geometry	Klassen zur Berechnung (Point, Bounds, Polyline) oder Darstellung (LineString, LinearRing, Polygon) von geometrischen Formen.
overlay	Overlays können an die Karte übergeben werden. Die Karte rendert diese dann.
parser	Ein GPX und ein KML Parser.
tilesource	Vorgefertigte JSON Objekte für diverse Tilesources, die anstatt oder zusätzlich des default Tilesource "OpenStreetmap" angezeigt werden können.
ui	Funktionalität zuR Interaktion des Users mit der Karte.
util	Hilfsklasse(n). Enthält Klassen für AJAX-Requests und Geo-Lokalisierung.
vector	Funktionalität zum Zeichnen von Vektoren.
vector.renderer	Canvas, SVG und VML Renderer für unterschiedliche Browser.

Tabelle 10.1: Package Beschreibung

10.3 PACKAGE- UND KLASSENDIAGRAMME

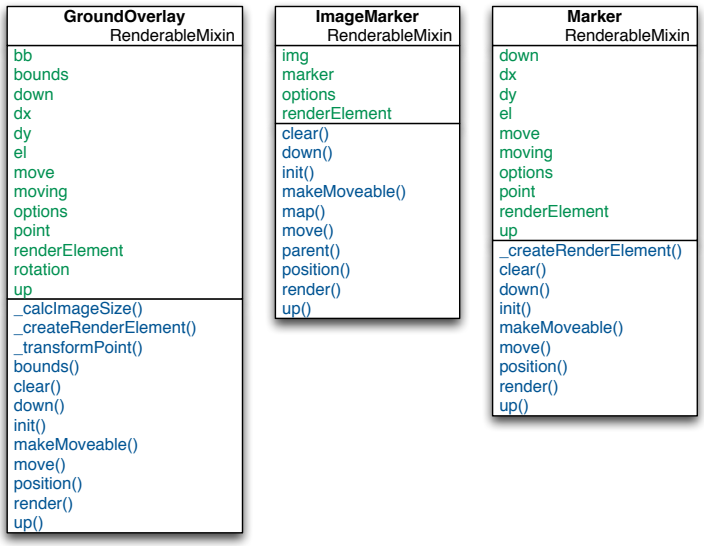
10.3.1 Package Base

Abbildung 10.4: Klassendiagramm Package: Base

Die Klasse 'Map' besteht aus sehr vielen Properties und Methoden. Für eine bessere Darstellung wurde ein Teil ausgelassen.

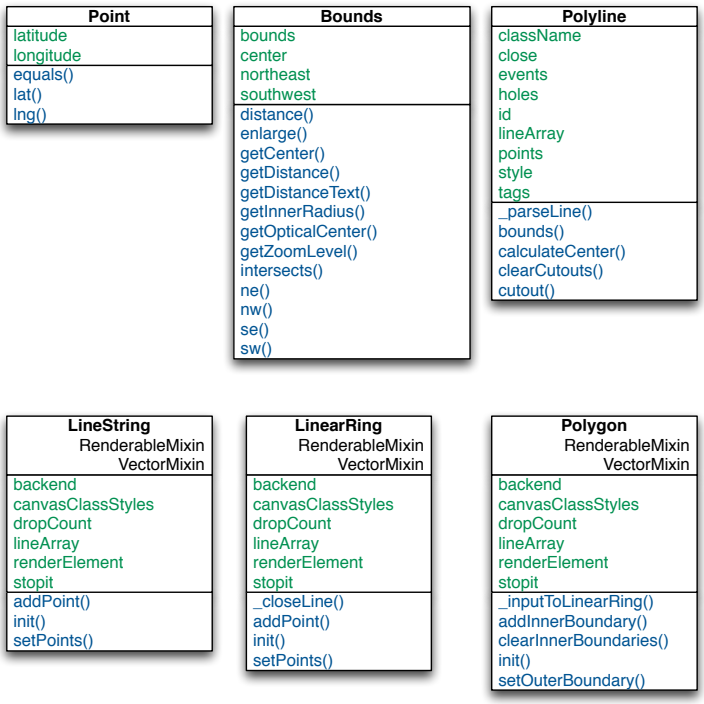
10.3.2 Package Displayable

Abbildung 10.5: Klassendiagramm Package: Displayable



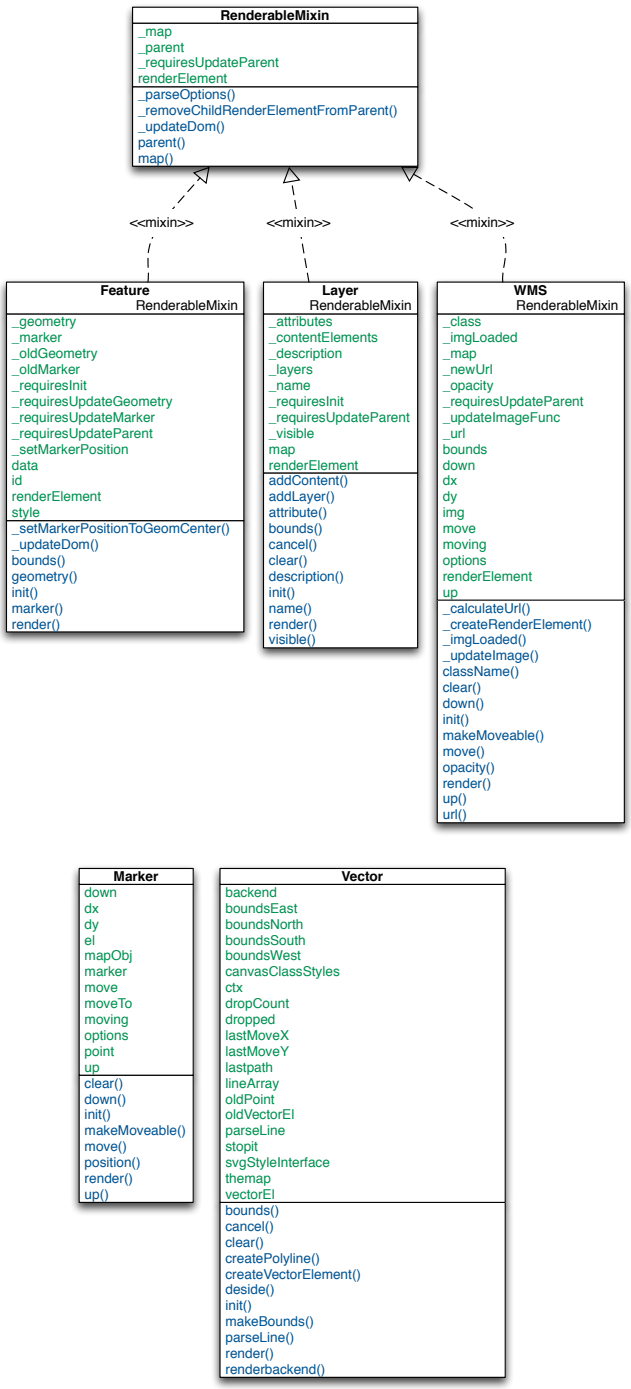
10.3.3 Package Geometry

Abbildung 10.6: Klassendiagramm Package: Geometry



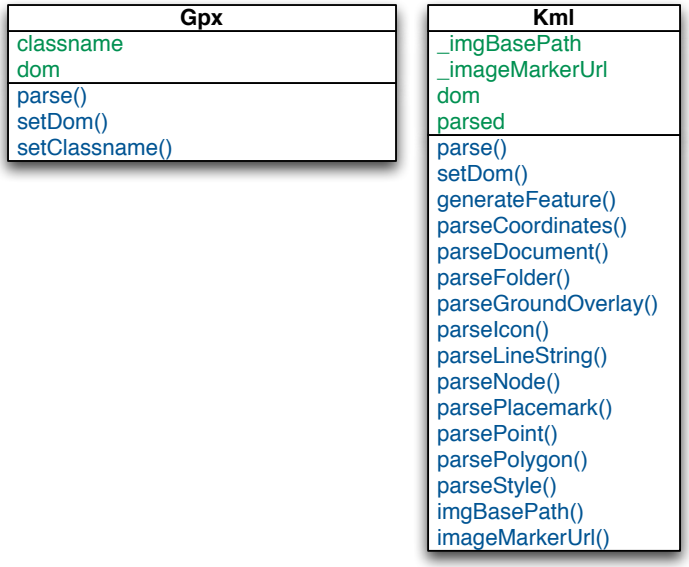
10.3.4 Package Overlay

Abbildung 10.7: Klassendiagramm Package: Overlay



10.3.5 Package Parser

Abbildung 10.8: Klassendiagramm Package: Parser

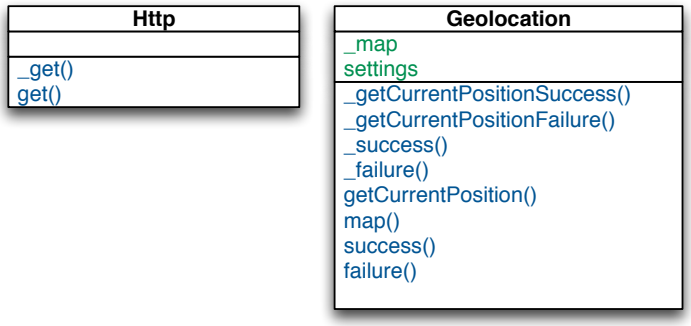


10.3.6 Package UI

Abbildung 10.9: Klassendiagramm Package: UI

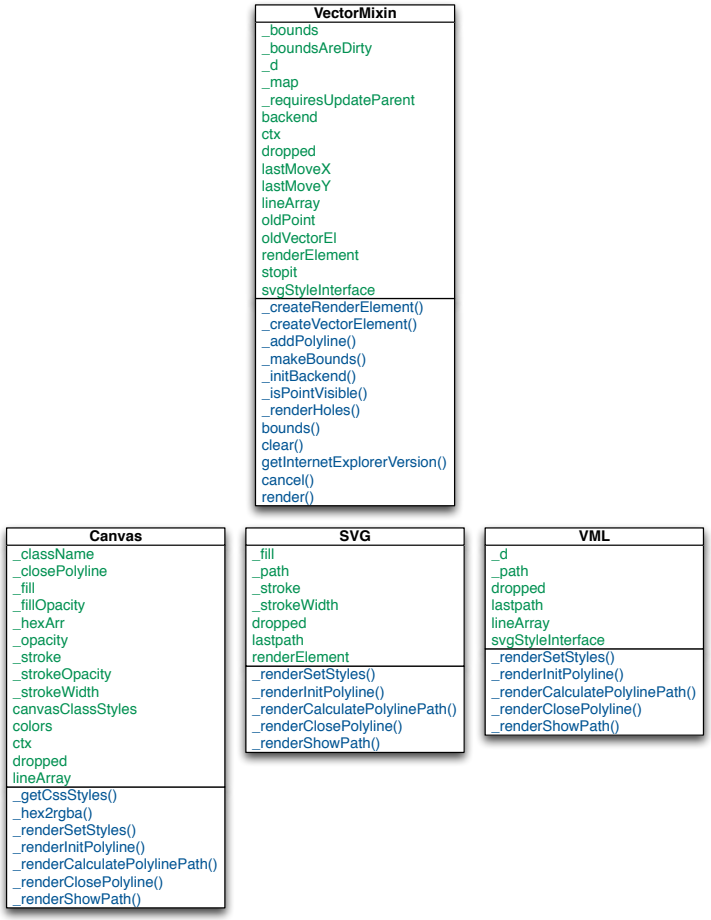
10.3.7 Package Util

Abbildung 10.10: Klassendiagramm Package: Util



10.3.8 Package Vector inkl. Subpackage Renderer

Abbildung 10.11: Klassendiagramm Package: Vector & Vector.Renderer



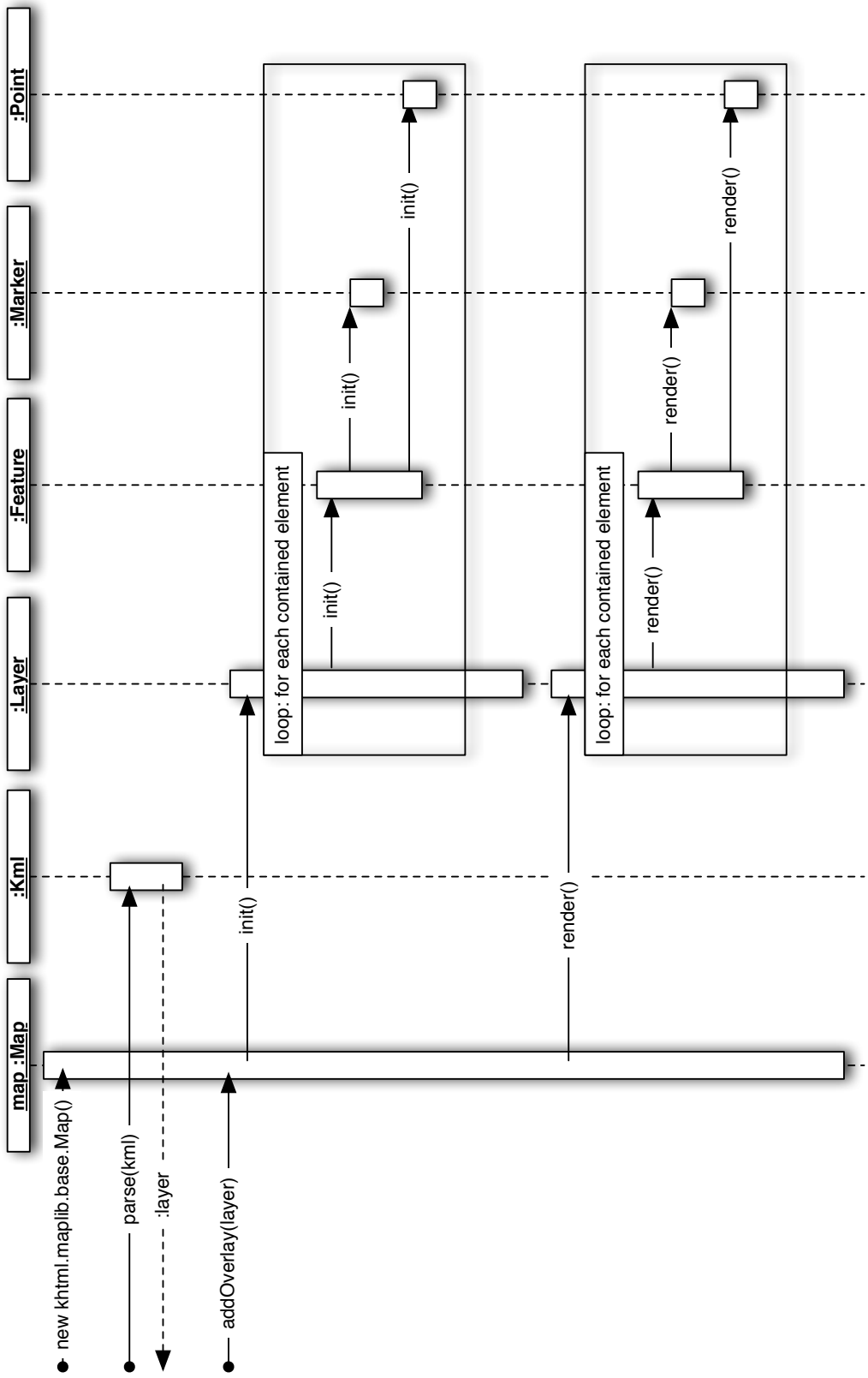
10.4 SEQUENZDIAGRAMME

10.4.1 KML Darstellen

Erläuterung des Sequenz-Diagramms.

- KML wird als String an die 'parse()' Methode übergeben. Diese liefert als Resultat ein 'Layers' Objekt zurück.
- 'Layers' implementiert das 'Overlay'-Interface und kann an 'addOverlay()' übergeben werden
- 1. Schritt: per 'init()' werden alle Default-Werte und Vater-Kind Verknüpfungen gesetzt
- 2. Schritt: per 'render()' werden die Elemente ins DOM Eingefügt und die Darstellung upgedatet.
- Beide Schritte werden durch alle Layer, Features, usw. propagiert.
- ':Marker' und ':Point' sind in Feature enthalten. Statt dem ':Point' könnten auch ein anderes Geometry Objekt (z.B. Polygon) enthalten sein.

Abbildung 10.12: KML Darstellen



11 IMPLEMENTATION

Im Folgenden werden wichtige konkrete Klassen erläutert und der Zusammenhang innerhalb der Khtmlib der einzelnen Klassen aufgezeigt.

Dabei ist es wichtig, die grobe Struktur der Bibliothek und einige Konzepte zu kennen. Diese werden vorab erklärt.

11.1 ÜBERBLICK ÜBER GROBE STRUKTUREN

Die Basisfunktionalität ist im Package 'base' zusammengefasst. Die eigentliche Karte wird dabei durch die Map-Klasse dargestellt.

Um verschiedene Datenformate auf der Karte zur Anzeige bringen zu können wird für jedes unterstützte Format eine Klasse im Paket Parser verwendet. Derzeit können GPX- und KML-Daten geparst und dargestellt werden.

Der Parser erstellt je nach Dateninhalt Layer-Objekte, welche Features enthalten und auf der Karte dargestellt werden können.

Die Features haben eine Geometrische Form (d. h. verwenden ein Objekt aus dem Paket 'geometry') und enthalten ein Objekt von einer Klasse aus dem Paket 'displayable'. Ebenfalls soll dem Feature in Zukunft ein Styleobjekt zugeordnet werden können. Dies ist derzeit aber noch nicht implementiert.

Unabhängig von Objekten/Klassen existieren Helper-Methoden, welche von einigen der Klassen verwendet werden, um den Code in den Klassen selbst kürzer und besser verständlich zu gestalten.

Mit den Klassen im Paket 'tilesource' kann die Kartenquelle des Basislayers gewählt werden. Sie können statt als Basislayer auch als zusätzlichen Kartenlayer verwendet werden. Diese Klassen bestehen aus der Methode 'src' welche die URL zum benötigten Kachel berechnet und zurück liefert, sowie Properties die den maximalen und minimalen Zoomlevel des Tilesources definieren.

11.2 VERERBUNG ÜBER MIXIN-KONZEPT

Als Strategie für die Wiederverwendung von Quellcode wird das Mixin-Pattern eingesetzt. Dabei kopiert das 'erbende' Objekt, die benötigten Methoden vom Mixin-Objekt in das eigene Objekt. Eine gute Erklärung zur Verwendung von Mixins in Javascript bietet[2].

Mit diesem Vorgehen wird auch Mehrfachvererbung möglich. Für das Kopieren wird die Methode khtml.maplib.base.helpers.mixin(target, source, props) verwendet.

Listing 11.1: Mixin Beispiel

```
khtml.maplib.base.helpers.mixin(
 khtml.maplib.displayable.GroundOverlay.prototype,
 khtml.maplib.overlay.RenderableMixin,
 new Array("_removeChildRenderElementFromParent", "
    _parseOptions", "parent",
 "map")
);
```

Im Beispielcode werden konkret von 'khtml.maplib.overlay.RenderableMixin' die Methoden welche im Array spezifiziert sind in den Prototype der Klasse 'khtml.maplib.displayable.GroundOverlay' kopiert. Lässt man den Array weg, werden alle Methoden und Properties von der Quelle zum Ziel kopiert.

11.3 STEUERUNG DER KARTE ÜBER DIE TASTATUR

Im File 'js/ui/Keyboard.js' wird die Mapklasse erweitert. Dadurch wird es möglich, die Karte mit der Tastatur zu steuern. So kann das Panning der Karte mit den Pfeiltasten sowie das Zoomen mit + und - erreicht werden.

11.4 ERLÄUTERUNGEN WICHTIGER KONKRETER KLASSEN

Eine detaillierte Dokumentation der einzelnen öffentlichen Methoden der Klassen liegt in Form einer API-Dokumentation vor und wird direkt aus den im Code angemerkten Kommentaren erzeugt. Dieses Kapitel soll auf wenigen Seiten einen Überblick geben.

11.4.1 khtml.maplib.base.Map

Die Map-Klasse repräsentiert die Karte in der Webseite. Dabei ist ein Objekt der Klasse eine Karte auf der Webseite. Es gibt keine Einschränkungen, wie viele Karten auf einer HTML-Seite angezeigt werden können.

Es wurden bewusst keine Anpassungen an der Map Klasse gemacht. Mit Ausnahme von Bugfixes und Adaptionen an die Erweiterungen. Dies deshalb weil die Klasse stark optimiert ist und verhindert werden sollte, dass versehentlich Optimierungen verloren gehen. Ausserdem wäre danach ein viel aufwendigeres Testen auf allen möglichen Geräten (Android, iPhone, usw.) nötig gewesen.

In der Map-Klasse sind Funktionalitäten wie zoomen, panning aber auch gezieltes Setzen einer gewünschten Position und Zoomstufe implementiert.

Listing 11.2: Karte einbinden

```
<html>
 <head>
  <title>Karte</title>
  <script type="text/javascript" src="khtml_all.js"> </
      script>
 </head>
 <body>
  <div id="meineKarte"></div>
  <script type="text/javascript">
   function init(){
        // Create Map Object
        var map=new khtml.maplib.base.Map("meineKarte")
            ;
        // Define Map Center
        var center = new khtml.maplib.geometry.Point
            (47.22331159092443,8.817104609290709);
        // Point map to "center" and adjust zoomlevel
        map.centerAndZoom(center,17);
   }
   init();
  </script>
 </body>
</html>
```

Im Codebeispiel wird ein Kartenobjekt erzeugt und im Tag mit der Id 'meineKarte' angezeigt. *Wichtig:* die Karte wird

erst angezeigt wenn ein Aufruf der Methode 'centerAndZoom()' stattgefunden hat.

Die Map Klasse erstellt als Basis-Layer einen Kachellayer. Das generieren der URL für die Kacheln kann überschrieben werden.

Pro Zoomstufe wird im HTML ein eigener DIV erzeugt. Im entsprechenden zoomstufen DIV, werden die Kacheln plaziert.

Weiterhin können über sogenannte Overlays, Elemente auf der Karte plaziert werden. Diese werden bei Darstellungsänderungen (zoom, panning, usw.) notifiziert und können ihre eigene Darstellung updaten.

11.4.2 khtml.maplib.base.Log

Diese Klasse ermöglicht das Logging der Funktionsweise der Khtmlib-Bibliothek und ist nur für die Entwicklung an der Bibliothek interessant. *Es sollten unbedingt diese Logging Methoden verwendet werden.* Im Gegensatz zu direktem Logging mit Firebug, kommt es zu keinen Exceptions in Browsern die kein Firebug installiert haben, wenn einmal ein Log Statement vergessen wurde auszukommentieren. Das Khtmlib Logging kann ausserdem komplett deaktiviert werden.

Informationen zur Anwendung finden sich im Entwicklerhandbuch Unterabschnitt 15.2.2 (S. 115).

11.4.3 khtml.maplib.geometry.Bounds

Bounds repräsentiert die Boundingbox der Karte oder eines anderen Objektes. Bei der Karte repräsentiert die Boundingbox den angezeigten Kartenausschnitt. Bei anderen Objekten (Marker, Polygon) liefert es die Koordinaten des kleinsten Rechteckes welches dieses Objekt einfassen kann. Die Boundingbox ist immer rechteckig und wird durch die südwestliche und nordöstliche Ecke des Rechtecks definiert.

11.4.4 khtml.maplib.parser.GPX

Diese Klasse nimmt als Input einen String im GPX-Format. Es werden die Tag's 'trkseg' und 'trkpt' unterstützt. Der Parser liefert ein 'khtml.maplib.overlay.Vector' Objekt zurück, welches pro Tracksegment ('trkseg') eine Polyline bestehend aus dessen Trackpunkten ('trkpt') enthält.

Die ursprüngliche Implementation hat das GPX Objekt selbst als 'Overlay' auf die Karte gelegt. Da dies nicht nötig ist und um

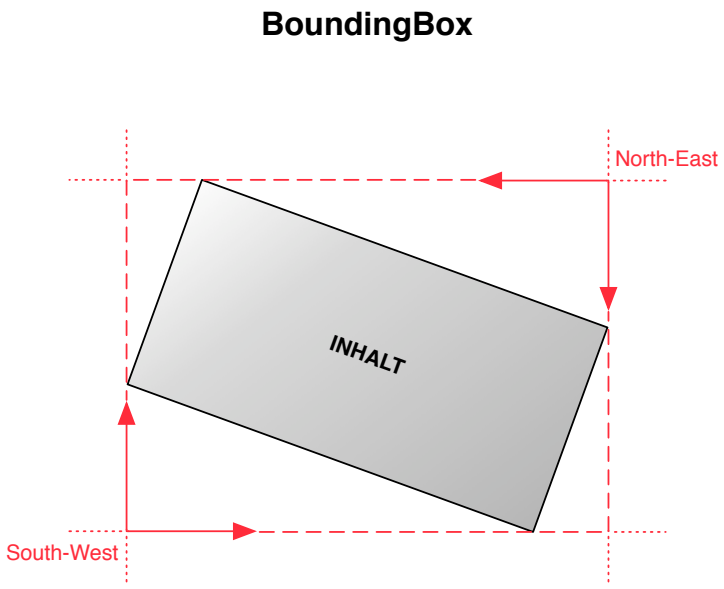


Abbildung 11.1: Boundingbox

die Komplexität zu reduzieren, liefert der GPX-Parser nun ein 'khtml.maplib.overlay.Vector' Objekt zurück.

11.4.5 khtml.maplib.parser.KML

Die KML-Klasse ist für das Parsen von KML-Daten zuständig. Als Input wird ein String im KML-Format erwartet. Zuerst wird aus dem String ein DOM Baum erzeugt. Der Parser daraus Feature und Layer Objekte und gibt den obersten Layer als Return-Wert zurück.

Diese können direkt dem Map-Objekt mit der Methode addOverlays(rückgabewertausparsefunktion) angehängt bzw. auf der Karte dargestellt werden.

Listing 11.3: KML Parser

```
kmlParser = new khtml.maplib.parser.Kml(data);
var overlays = kmlParser.parse();
map.addOverlays(overlays);
```

Die Variable 'data' enthält im Beispiel die KML-Daten. Die Variable map ist das Map-Objekt.

Beim Parse-Vorgang durchläuft der Parser den gesamten DOM-Baum. Dadurch wird sichergestellt, dass die Hierarchie korrekt

in Layer Objekte abgebildet werden kann und alle Elemente verarbeitet werden.

11.4.6 khtml.maplib.displayable.Marker

Mit dem Markerobjekt kann ein HTML-Element an einer frei wählbaren georeferenzierten Stelle in der Karte Angezeigt werden. Die Klasse kann somit für die Darstellung von ganz verschiedenen Dingen (z. B. Labels, Bilder oder auch Popups) verwendet werden.

Die Position wird innerhalb des Map-Containers über CSS-Positionierung gesteuert. Beim Kartenverschieben wird der Marker mitverschoben.

11.4.7 khtml.maplib.displayable.ImageMarker

Die Klasse ImageMarker verwendet die Klasse khtml.maplib.displayable.Marker. Damit kann direkt ein Bild als Marker auf der Karte platziert werden. Einsatzzweck ist z.B. für POI-Icons. Damit muss nicht jedesmal das benötigte HTML für ein Bild-Marker manuell erstellt werden.

11.4.8 khtml.maplib.displayable.GroundOverlay

Mit der Klasse GroundOverlay ist es möglich, Bilder auf dem Kartengrund zu fixieren. Im Gegensatz zu khtml.maplib.displayable.ImageMarker werden die Groundoverlay-Bilder mit der Karte skaliert wenn diese gezoomt wird.

Die übergebenen Bilder können ausserdem mit der Option 'rotation' nach belieben gedreht werden. Dazu kommt die Bibliothek jQuery mit dem Plugin jQueryRotate zum Einsatz. Diese beiden Bibliotheken müssen im Header der Seite mit folgendem Code eingebunden werden.

```
<script type="text/javascript" src="path/to/lib/jquery
    -1.5.1.min.js"> </script>
<script type="text/javascript" src="path/to/lib/
    jQueryRotate.2.1.js"> </script>
```

Mit new khtml.maplib.displayable.GroundOverlay(myBB, myIcon, options); wird ein Groundoverlay mit der Boundingbox myBB und dem Bild myIcon erstellt. Will man eine Drehung des Bildes bewirken muss options.rotation eine Zahl vom typ Float mit dem Winkel enthalten.

11.4.9 khtml.maplib.overlay.Layer

Die Klasse Layer wird verwendet um Geodaten auf der Karte abzubilden. Vergleicht man ein Layerobjekt mit einem Filesystem, so gleicht das Layer-Objekt einem konkreten Ordner.

Das Layer-Objekt enthält für gewöhnlich eine Liste von Feature-Objekten (ähnlich wie Files im Dateisystem) sowie eine Liste von Layer-Objekten. Damit kann eine Verschachtelung erreicht bzw. in den Geodaten enthaltene hierarchische Informationen abgebildet werden.

Layer-Objekte können mit der Methode 'visible()' sichtbar oder unsichtbar gemacht werden.

11.4.10 khtml.maplib.overlay.Feature

Features haben eine konkrete Geometrie (also ein Objekt aus dem Paket khtml.maplib.geometry), optional ein Objekt vom Typ khtml.maplib.displayable sowie Style-Objekte (zurzeit noch nicht implementiert).

Features sind konkrete Dinge (Marker, Groundoverlay, Linien, Flächen, etc.) welche auf der Karte dargestellt werden. Somit können einerseits Rasterdaten (GroundOverlay, Marker) oder Vektordaten (Linien, Flächen, etc.) dargstellt werden. Features ihrerseits sind in Layer-Objekten abgelegt. Die Layer werden der Karte zugeordnet und damit werden schlussendlich die Features auf der Karte angezeigt. Vom Konzept her ist ein Feature-Objekt mit dem File in einem Dateisystem zu vergleichen. Ordner sind die Layer-Objekte.

Bei einem Feature können Zusatzinformationen abgelegt werden. So werden z.B. die Title und Description Texte des KML-Placemarks im 'data' Property des Features abgelegt.

11.4.11 khtml.maplib.overlay.WMS

Mit einem Objekt dieser Klasse kann eine WMS-Quelle als Kartenoverlay in die Karte eingebunden werden. Im Gegensatz zu dem Basislayer, der aus mehreren Kachelbildern besteht, wird von der WMS-Quelle genau 1 Bild von der Grösse der Karte abgerufen und über der Karte positioniert.

Bei Kartenaktionen (Zoom, Pan[1]) wird nach dem Event, die Karte aktualisiert, sprich ein neues Bild vom WMS-Server, mit dem neuen Kartenausschnitt angefordert.

1 verschieben

Es wurde speziell darauf geachtet, während dem Pan- oder Zoomvorgang keine Bilder für die Zwischenschritte vom WMS Server anzufordern. Dies bedeutet nur eine unnötige Last für den WMS-Server. Die Aktion ist ausserdem schneller beendet, als Bilder vom WMS-Server gerendert und über Netzwerk nachgeladen werden können.

11.4.12 khtml.maplib.vector.VectorMixin

Diese Klasse erledigt das Zeichnen von Linien und Flächen. Die Methoden für das Rendering sind abstrakt. Je nach eingesetztem Browser wird von VectorMixin das passende Backend bestimmt. Aufgrunddessen lädt die Geometrieklasse (LineString / LinearRing / Polygon) das passende Vektor-Render-Mixin. Das Vektor-Render-Mixin ist eines aus: 'khtml.maplib.vector.renderer.Canvas' / 'khtml.maplib.vector.renderer.SVG' / 'khtml.maplib.vector.renderer.VML'.

12 TESTING

12.1 AUTOMATISCHE TESTVERFAHREN

12.1.1 Testframework

Für Unittesting verwenden wir Jasmine[1], ein Javascript BDD[2] Framework.

12.1.2 Testabdeckung

BESTEHENDER CODE Wir haben für alle bestehenden Klassen Testcases geschrieben. Aus Zeitgründen haben diese Klassen unterschiedlich grosse Testabdeckung. Das Ziel für alle bestehenden Klassen Tests zu schreiben und deren wichtigste Methoden abzudecken, wurde erreicht.

NEUER CODE Für neu erstellten Code soll eine Testabdeckung von 100% erreicht werden.

12.1.3 Testmethodik

Im Sinne eines Blackblox-Testing werden die öffentlichen Methoden getestet. (Private Methoden & Properties beginnen mit einem '_' im Namen.) Wo sinnvoll werden auch interne Zustände der Objekte geprüft, sowie getestet ob bestimmte (interne) Methoden aufgerufen wurden.

Wir verwenden kein komplettes BDD. Wir verwenden einen Mix aus traditionellen Testcases und 'behaviour'-basierten Testcases. Zudem nutzen wir die Möglichkeit in Worten zu beschreiben was der Testcase prüfen soll, sowie die von Jasmine gebotenen Inspection-Möglichkeiten.

12.1.4 Tests ausführen

Die Testcases können im Browser ausgeführt werden. Wichtig: Khtmlib muss dazu auf einem Webserver abgelegt werden. An-

1 https://github.com/pivotal/jasmine
2 http://dannorth.net/introducing-bdd/

schliessend können die Tests über die URL `http://meinwebserver/pfad-zu-khtmlib/testcases/unittesting/` aufgerufen werden.

12.1.5 Browserkompatibilität

Resultate der Jasmine Testsuite in folgenden Browsern:

BROWSER	OS	RESULTAT
IE8	Windows 7	Fails
Firefox 4	OSX 10.5	Grün
Safari 4	OSX 10.5	Grün
Chrome 11	OSX 10.5	Grün
Webkit 3.1	Android 2.21 (HTC Desire Z)	Grün
Safari	iPhone 4	Grün

Tabelle 12.1: Browserkompatibilität Testcases

IE8 TEST Hier schlägt der XML-Parser Test fehl. Da Internet Explorer Support eine optionale Anforderung ist, darf dieser Test fehlschlagen, siehe auch Unterabschnitt 6.2.4 (S. 51).

12.2 MANUELLE TESTVERFAHREN

12.2.1 Manuelle Tests

Es gibt zwei Ordner mit manuellen Testcases. Diese Testcases müssen ebenfalls über einen Webbrowser ausgeführt werden. Die URL's sind: `http://meinwebserver/pfad-zu-khtmlib/testcases/basics/` und `http://meinwebserver/pfad-zu-khtmlib/testcases/vector/`.

Diese Testcases wurden für die Implementation der Features verwendet. Ausserdem dienen Sie auch als Demonstration und Anwendungsbeispiele.

12.2.2 Systemtest

Manuelle Tests beziehen sich auf die Testfiles im Ordner 'testcases/basics/' bzw. 'testcases/vector/'. Tests im Webbrowser wurden mit Firefox 4.0 auf OSX und Android Webkit auf HTC Desire Z durchgeführt.

TEST	STATUS
MANUELLE TESTS	
Darstellung LineString Tests	OK
Darstellung LinearRing Tests	OK
Darstellung Polygon Tests	OK
Darstellung GroundOverlay Tests	OK
Darstellung KML Tests	OK
Darstellung WMS Tests	OK
WEITERE TESTS	
UnitTests	OK
Build funktioniert?	OK

Tabelle 12.2: Systemtest (16.06.2011)

13 PROJEKTMANAGEMENT

13.1 PROJEKTMANAGEMENT MITHILFE VON REDMINE

Die Software Redmine bietet alles was nötig ist um ein Softwareentwicklungsprojekt erfolgreich durchzuführen. Es ist ein optimales Zusammenspiel zwischen SVN und dem Ticketingsystem möglich. Redmine bietet ein Ticketingsystem, Integration in unser Versionsverwaltungsystem (Subversion), Wiki, E-Mail benachrichtigungen und vieles mehr. Ausserdem existierte bereits eine Installation von Redmine auf dem Server von Florian Hengartner, welche wir benutzen konnten.

13.2 PROJEKTMETHODIK (SCRUM)

Wir realisierten unser Projekt nach den Grundlagen von Scrum. Scrum wählten wir deshalb, weil es eine Agile Softwareengeneering-Methode ist und eine Agile Entwicklungsmethode eine Vorgabe für unser Projekt war. Da zu Projektstart auch noch nicht alle Ziele bekannt waren, sorgt Scrum für die nötige Flexibilität auf veränderte Umstände reagieren zu können.

Da bei Scrum ein Team üblicherweise aus 5-9 Personen besteht und wir zu zweit ein Projekt entwickeln, mussten wir Scrum noch weiter auf unsere Bedürfnisse anpassen.

Zu Anfang nutzten wir ein spezielles Scrum-Plugin. Fehler in diesem Plugin veranlassten uns die Scrum-Methode manuell in Redmine umzusetzen. User Stories wurden als Tickets erfasst, priorisiert und einem Sprint-Backlog oder dem Product-Backlog zugewiesen. Dies hat gut funktioniert.

13.2.1 Geplante Adaption von Scrum

13.2.1.1 *Rollen*

Product Owner ist Stefan Keller. Das Scrum Team besteht aus Stefan Kemper und Florian Hengartner. Die Rolle des Scrum Masters übernimmt Florian Hengartner.

13.2.1.2 *Zyklen*

Weil das Projekt nur ein Semester lange dauert und die gewünschten User-Stories stark ändern können, entschieden wir uns für eine kurze Sprintphase von jeweils zwei Wochen.

Für jeden Sprint wird in Redmine ein Kategorie erstellt. Alle Tickets die zum Sprint gehören werden dieser Kategorie zugeordnet. Ebenfalls besteht ein allgemeine Kategorie für das 'Product Backlog'.

13.2.1.3 *Sitzungen*

Die Planungstreffen für die Sprints werden mit dem Product Owner (Stefan Keller) direkt in die wöchentliche Sitzung integriert.

Auf physische Daily Scrum Meetings wird verzichtet. Anstelle der Daily Scrum Meetings treten regelmässige Kommunikation per Mail, Wiki und Skype. Desweiteren wird über die Kommentare bei den Tickets und den Checkins ins Subversion Respository zu kommuniziert.

13.2.2 Erfolgte Adaption von Scrum

Die geplante Scrum-Adaption wurde mit folgenden Anpassungen umgesetzt.

13.2.2.1 *Zyklen*

Aufgrund der allwöchentlich stattfindenden Projektsitzung, wurde die Dauer eines Sprints auf sieben Tage reduziert.

13.2.2.2 *Sitzungen*

Oft wurde zeitlich getrennt Entwickelt (z.B. Florian Hengartner von Mo-Di und Stefan Kemper von Do-Fr). So konnte anstelle täglicher Scrum Meetings jeweils am Ende eines solchen Abschnittes eine Übergabe durchgeführt werden.

Die Verwendung des Ticketsystems hat sich bewährt und die Zuteilung und Planung von Aufgaben sehr erleichert.

13.2.2.3 *Artefakte*

Wir verwenden folgende Artefakte:

- Product Backlog
- Sprint Backlog

- Gantt Chart (als Alternative zur klassischen Burndown chart)

Da nach dem Wegfall des Scrum Plugins die Burndown Chart ebenfalls wegfiel, verwendeten wir als Alternative die Gantt Chart, welche den Arbeitsstand auch sehr gut aufzeigt.

13.2.2.4 *Umgang mit Bugs*

Da das Backlog während dem Sprint unveränderlich ist (Vorschläge zur möglichen Handhabung: `http://blog.mountaingoatsoftware.com/bugs-on-the-product-backlog`, `http://ericlefevre.net/wordpress/2008/02/03/how-to-handle-bugs-in-the-sprint-backlog/`) haben wir uns für das folgende Vorgehen entschieden: Tauchen die Bugs bei Features vom aktuellen Sprint auf, so werden die dazu gehörenden Tickets ins Sprint Backlog eingetragen. Sind es jedoch Bugs von anderen (bestehenden) Features, dann werden die auf das Product Backlog gelegt.

13.3 VERSIONSKONTROLLE (SVN! (SVN!))

Als Versionskontrolle wird SVN verwendet. Es wird von Redmine perfekt integriert: Commit Nachrichten werden beim zugehören Ticket angezeigt und es ist ein Repository Browser vorhanden. SVN deckt sämtliche unserer Anforderungen an eine Versionskontrolle ab und ist bereits auf der verfügbaren Serverinfrastruktur installiert.

13.4 REVIEWS

Bei den wöchtenlichen Team-Sitzungen (nicht zu verwechseln mit den Projekt-Sitzungen mit Herr Keller), wurde gemeinsam die letzte Woche besprochen. Aufgetretene Probleme wurden diskutiert. Code-Reviews wurden in unregelmässigen Abständen durchgeführt. Da nicht genügend Zeit vorhanden war alle geleisteten Arbeiten gemeinsam durchzugehen, wurde jeweils nur ein Ausschnitt des Codes gewählt.

13.5 RISIKEN

In der folgenden Risikoanalyse listen wir die möglichen Risiken auf und bewerten sie. Aufgrund der Risiken wird zusätzliche Zeit bei der Planung der einzelnen Arbeiten eingerechnet.

Was		Schaden (in h)	Eintrittsw.	gew. Schaden	Prio.
Risiko:	Zusammenarbeit mit Maintainer	40	30%	12	mittel
Auswirkungen:	Verzögerungen, Codeänderungen während Arbeit				
Massnahmen:	Regelmässige Kommunikation mit Maintainer				
Risiko:	Probleme in bestehendem Code	100	50%	50	hoch
Auswirkungen:	Verzögerungen, Refactoring notwendig				
Massnahmen:	Refactoring durchführen				
Risiko:	Performance Probleme	100	20%	20	hoch
Auswirkungen:	Maintainer übernimmt Erweiterungen nicht				
Massnahmen:	Performance Tests, Reserve für Optimierungen				
Risiko:	Ausfall Redmine	40	5%	2	niedrig
Auswirkungen:	Führen von Arbeitszeit unmöglich, Datenverlust bei Zeitaufwand				
Massnahmen:	Backup der Infrastruktur				
Total:				84	h

Abbildung 13.1: Risiko-Analyse

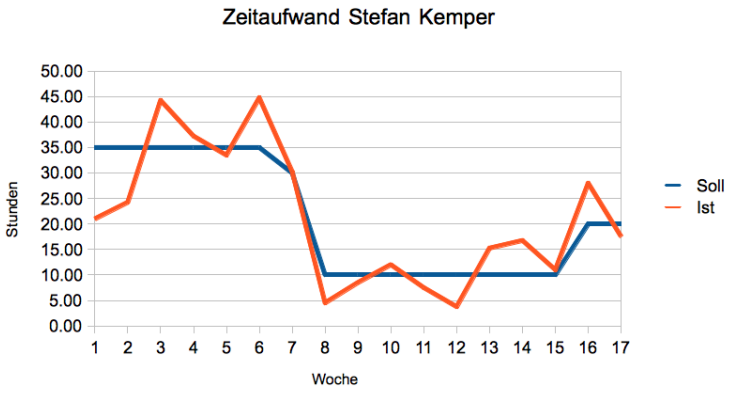
14 PROJEKTMONITORING

14.1 SOLL-IST-ZEIT-VERGLEICH

14.1.1 Stefan Kemper

Stefan Kemper hat von den 360 Stunden, welche zur Verfügung standen, 360 Stunden aufgewendet (100%). In folgendem Diagramm ist die geplante Zeit (blau) der tatsächlich eingesetzten Zeit (rot) gegenüber gestellt.

Abbildung 14.1: Zeitvergleich Stefan Kemper



Der Planung ist zu entnehmen, dass Stefan Kemper zu Beginn des Projektes mehr Zeit aufwendete, danach eine Phase hatte wo er weniger Zeit investierte. Gegen Schluss der Arbeit plante er einen Schlussspurt. Grund für diese Planung sind die Arbeitszeiten, welche durch den Arbeitgeber von Herrn Kemper gegeben waren.

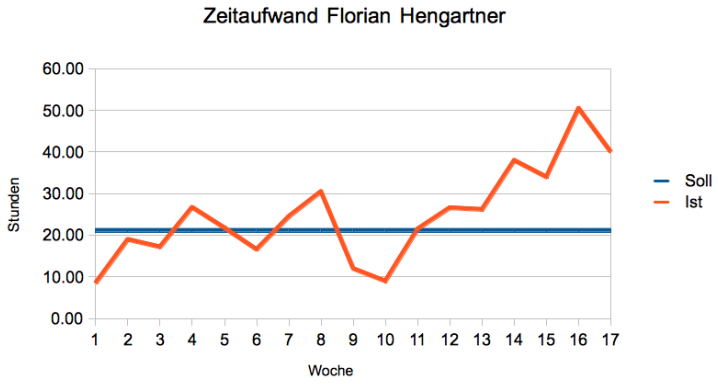
Er konnte den vorweg gemachten Plan meistens relativ gut einhalten. Starke Schwankungen einzelner Wochen sind auf die von seinem Arbeitgeber geforderten unregelmässigen Arbeitszeiten zurück zu führen.

14.1.2 Florian Hengartner

Den geforderten 360 Stunden Aufwand stehen 423 Stunden geleistete Arbeit gegenüber. In folgendem Diagramm ist die geplante

Zeit (blau) der tatsächlich eingesetzten Zeit (rot) gegenüber gestellt.

Abbildung 14.2: Zeitvergleich Florian Hengartner



Ganz gut sieht man den Einbruch in der Oster Woche und den Anstieg gegen Ende der Projektarbeit. Die grösseren Arbeitszeiten gegen Ende des Projektes waren zum Teil geplant, als Kompensation von Stefan Kemper, der dann weniger Zeit hatte.

14.2 SITZUNGSPROTOKOLLE (NUR AUF CD!)

Befinden sich auf der CD im Ordner 'Dokumentation/Sitzungsprotokolle'.

14.3 CODESTATISTIK

Die Statistik wurde mit CLOC[1] generiert.

1 http://cloc.sourceforge.net/

METRIK	RESULTAT
CORE CODE	
Dateien	39
Leerzeilen	990
Kommentare	3424
Code	5687
Klassen (manuell gezählt)	30
CODE INKL. UNITTESTS	
Dateien	71
Leerzeilen	1620
Kommentare	4021
Code	7994

Tabelle 14.1: Metrik

15 SOFTWAREDOKUMENTATION

15.1 BENUTZERHANDBUCH

Das Benutzerhandbuch befindet sich im Projekt im Ordner "doc/tutorial/". Es muss über einen Webserver betrachtet werden, da es ein PHP Script (doc/tutorial/index.php) ist.

Für die Dauer des Projektes ist es unter `http://sinv-56012.edu.hsr.ch/code_trunk/khtmlib_refactored/doc/tutorial` einsehbar. Dieser Server wird von der HSR nur für die Dauer der Bachelorarbeit zur Verfügung gestellt.

ORDNER	BESCHREIBUNG
img	Bilder
build	Diesen Ordner kann der Endbenutzer kopieren und für eine vollständige Khtmlib installation.
jQueryHelper	jQuery[1] Javascript-Bibliothek
testcases	Unittesting und manuelle Testcases
css	CSS File(s)
js	Khtmlib Javascript Code
tools	Werkzeuge zur Dokumentation, Builderstellung und Javascript-Kompilation
doc	API- und Benutzerdokumentation

Tabelle 15.1: Ordnerstruktur

15.2 ENTWICKLERHANDBUCH

15.2.0.1 *Ordnerstruktur*

15.2.1 Tools

15.2.1.1 *API Dokumentation updaten*

Konsolen Kommando um die Javascript API Dokumentation frisch zu generieren:

```
# Hinweis: Dazu muss man im Hauptverzeichnis stehen.
make jsdoc
```

Zur Generierung der Dokumentation kommt das "JsDoc Toolkit"[2] zum Einsatz.

15.2.1.2 *Javascript kompilieren*

Konsolen Kommando um das Javascript frisch zu kompilieren:

```
# Hinweis: Dazu muss man im Hauptverzeichnis stehen.
make compile
```

Output:
'khtml_all.js': Wird aus allen Javascript Files generiert.
'khtml_min.js': Enthält die minimal nötigen Klassen (Map, Helpers, Log, Point, Bounds).

Wichtig: gibt es neue Javascript Files müssen diese in der Datei "tools/compile_js.sh" hinzugefügt werden.

Für das Kompilieren wird der "Closure-Compiler"[3] verwendet. Dadurch werden alle Javascript's in ein einziges File kombiniert und die Dateigrösse verkleinert.

15.2.1.3 *Build erstellen*

Diese Aktion generiert die API-Dokumentation und kompilierten Javascripts neu. Anschliessend werden alle nötigen Files in den "build" Ordner kopiert.

```
# Hinweis: Dazu muss man im Hauptverzeichnis stehen.
make build
```

15.2.2 Logging

Logging mit Firebug[4] ist sehr praktisch zum Debuggen oder Statusmeldungen anzeigen. Der Nachteil dabei ist, dass diese Logstatements im produktiven Code unbedingt entfernt werden

2 http://code.google.com/p/jsdoc-toolkit/
3 http://code.google.com/closure/compiler/
4 http://getfirebug.com/

müssen, da es sonst in Browsern ohne Firebug zu Exceptions kommen kann.

Deshalb sollte stattdessen die Khtmlib Logfunktionen verwendet werden. Diese sorgen dafür, dass es zu keinen Exceptions kommt. Beispiel für ein Logstatement:

Listing 15.1: Logging Beispiel

```
khtml.maplib.Log.log('Example Text');
```

Weitere Logfunktionen können in der API-Dokumentation nachgeschlagen werden.

Die Khtmlib Logfunktionen können global deaktiviert werden.

Listing 15.2: Logging de-/aktivieren

```
// deaktivieren
khtml.maplib.base.Log.disable();
// aktivieren
khtml.maplib.base.Log.disable();
```

15.2.3 Vector Renderer forcieren

Zum Entwickeln ist es ganz praktisch wenn man den Canvas Renderer auch im Desktop Browser ausprobieren kann. Dazu kann als Url-Parameter '?khtmlibrenderer=canvas' verwendet werden. Dies Funktioniert auch mit SVG oder VML. Liste möglicher URLs:

```
http://localhost/example.html?khtmlibrenderer=canvas
http://localhost/example.html?khtmlibrenderer=svg
http://localhost/example.html?khtmlibrenderer=vml

http://localhost/example.html?a=b&khtmlibrenderer=canvas
http://localhost/example.html?a=b&khtmlibrenderer=svg
http://localhost/example.html?a=b&khtmlibrenderer=vml
```

15.3 INSTALLATION

Siehe Benutzerhandbuch Abschnitt 15.1 auf Seite 113.

15.3.1 Für Entwickler

15.3.1.1 *Anforderungen*

Für die Entwicklung an der Khtmlib Blibiothek ist folgendes notwendig:

- Webserver mit PHP5 (für Testsuite)
- Unixsystem mit Bash für Kommandozeilen Tools (Unterabschnitt 15.2.1).

15.3.1.2 *Vorgehen*

Das gesamte Projekt auf einem Webserver ablegen. Nun kann bereits mit der Entwicklung begonnen werden.

Teil V

ANHANG

A PERSÖNLICHE BERICHTE

A.1 STEFAN KEMPER

A.1.1 Einleitung

Dieser Abschnitt enthält Erfahrungen und Rückblickende Gedanken von Stefan Kemper über die Bachelorarbeit zum Thema "Karten-Apps für Mobile Browser mit HTML5 und OpenStreetMap-in-a-Box". Für die Einarbeitung in OpenLayers und dem anschliessenden Vergleich mit Khtmlib wurde für die Einarbeitung in OpenLayers der Interaktive Lageplan der HSR verbessert. Anschliessend wurde die Khtmlib um Funktionalität erweitert. Die Erfahrungen sind im wesentlichen in diese beiden Bereiche (Interaktiver Lageplan & Khtmlib) gegliedert.

A.1.2 Interaktiver Lageplan

Die Einarbeitung in OpenLayers und der damit verbundenen Verbesserung des interaktiven Lageplans war sehr interessant. Es war sehr viel Neues zu lernen. Die einzelnen Schritte in der Verbesserung des Lageplans dauerten länger als erwartet. Dies hängt mit der doch sehr umfangreichen, komplexen und manchmal zu wenig detailiert dokumentierten OpenLayers-Library zusammen. Mit einem gewissen Grundwissen (welches mir am Anfang noch fehlte), findet man sich dann aber doch relativ gut in der vorhandenen Doku zurecht. Als Nachschlagewerk für Entwickler mit Erfahrungen im Map-Bereich ist diese super.

A.1.3 Khtmlib

Ab der sechsten Woche befassten wir uns mit der eigentlichen Hauptaufgabe und begannen Khtmlib zu überarbeiten und zu verbessern. Im Vergleich zu OpenLayers wird schnell klar, dass Khtmlib bislang eher schlecht dokumentiert und strukturiert ist. Auch automatisierte Tests der Bibliothek waren nicht vorhanden.

Erst mit dem Beginn der Arbeit an Khtmlib lernte ich die Dokumentation und den Funktionsumfang von OpenLayers sehr zu schätzen. Aufgrund des Codes bemerkten wir relativ schnell, dass hinter dem Khtmlib nur ein Entwickler (Hr. Zwischenbrug-

ger) treibende Kraft gewesen war und einfach seine Sicht der Dinge und nicht die Sicht eines grossen Entwicklerteams sich im Code wiederspiegelte.

So hat sich Florian Hengartner vor allem damit beschäftigt, Khtmlib erst zu refactoren. Ich schrieb vor allem Unittests zum bestehenden Code. Persönlich habe ich vor allem das Schreiben von Unittests mit der Jasmine-Testbibliothek als sehr wertvoll empfunden, da wir unsere Neuentwicklungen sowie das Refactoring jeweils automatisiert auf Fehler überprüfen konnten. Oft haben wir so Fehler entdeckt, welche sich eingeschlichen hatten.

Das Schreiben der Unittests war allerdings relativ aufwendig. Nach gewissen Code-Refactorings mussten auch die Unittests wieder angepasst werden. Sobald aber mehr wie eine Person am Code arbeitete, erlebte ich die Unittests als ein sinnvolles Werkzeug um zu testen, ob ich mit meinen Änderungen allenfalls den Code von jemand anderem versehentlich unbrauchbar gemacht habe. Meine persönliche Schlussfolgerung daraus ist, dass die Verwendung von Unittests bei grösseren und unüberblickbaren Projekten sehr zu empfehlen ist.

In der Weiterentwicklung von Khtmlib setzten wir uns vor allem für die Unterstützung von KML ein. Dabei unterstützt die Khtmlib Bibliothek nun auch gedrehte Groundoverlays und hat damit zumindest in einem Punkt gegen über OpenLayers gewisse Vorteile.

Meiner Meinung nach hat aber Khtmlib trotz unseren Entwicklungen einen schweren Stand gegenüber OpenLayers bestehen zu können. Vor allem weil OpenLayers gerade im Mobile-Sektor wieder am Aufholen ist gegenüber Khtmlib.

A.1.4 Fazit

Die Arbeit empfand ich als spannend und lehrreich. Ich habe gelernt wie man Landkarten im Web einsetzen kann. Gerade für zukünftige Webseiten mit Lageplan oder anderen interaktiven Landkarten konnte ich viel nützliches lernen.

Für mich ist jedoch fraglich, ob Khtmlib und damit unsere Erweiterungen gegenüber OpenLayers überlebensfähig sind. Ich hoffe es. Allerdings ist der Funktionsumfang von OpenLayers und die Community, welche OpenLayers weiter entwickelt wesentlich grösser als die von Khtmlib, weshalb es die kleine Bibliothek gegen den Giganten schwer haben könnte.

A.1.4.1 *Zeiteinteilung*

Die Zeiteinteilung war für mich eine grosse Herausforderung. Während meiner Bachelorarbeit war ich zu 20-40% beim BESJ und zu 80% beim Youth Computer Club Netzwerk angestellt. Von Februar bis April erhielt ich vom Youth Computer Club Netzwerk frei um mich voll der Bachelor-Arbeit zu widmen. Ab April arbeitete ich dann wieder regulär.

Somit teilte ich meine Zeit so ein, dass ich in den ersten drei Monaten meiner Arbeit 35 Stunden pro Woche arbeitete und danach nur noch 10-20 Stunden. Damit gelang es mir Studium und Beruf unter einen Hut zu bringen. Die zeitliche Belastung während der Bachelorarbeit war jedoch sehr hoch und ich musste private Dinge stark zurückstellen, was ich nicht immer als leicht empfand.

A.2 FLORIAN HENGARTNER

A.2.1 Interaktiver Lageplan

Die Arbeit am interaktiven Lageplan war eine gute Einführung in die GIS Thematik. Bisher hatte ich mit diesem Umfeld erst in der Datenbank 2 Vorlesung Kontakt. Es gab sehr viel zu Lernen, was sehr spannend war.

Über den interaktiven Lageplan haben wir praktische Erfahrungen mit OpenLayers sammeln können. Der Umfang von OpenLayers ist enorm, die Möglichkeiten die man damit hat sind Gewaltig. Aufgrund der Grösse und Komplexität war es Anfangs schwierig sich zurecht zu finden.

Bei der Überarbeitung des interaktiven Lageplans war für uns alles neu. Dementsprechend Langsam ging die Arbeit anfangs voran. Diese Arbeiten waren im späteren Verlauf der Projektarbeit sehr hilfreich, da ich nun alles Einordnen konnte.

Wir haben auf der bestehenden Basis des interaktiven Lageplans für unsere Verbesserungen weitergearbeitet. Im Nachhinein bin ich der Ansicht es wäre effektiver gewesen, wenn wir mit einer sauberen Basis neu angefangen und lediglich die relevanten Teil kopiert hätten.

A.2.2 Khtmlib

Der Einstieg bei Khtmlib präsentierte sich aus anderen Gründen als schwierig. Das Projekt ist nicht sehr gross. Jedoch ist der Code organisch gewachsen, unübersichtlich, schlecht kommentiert.

Ebenfalls fehlten eine Benutzerdokumentation, Entwicklerdokumentation, API Dokumentation und UnitTests. Verständlich, da es ein kleines 1-Mann Projekt ist. Mühsam für Entwickler wie uns. Als Hilfreich erwiesen sich die 'manuellen Testcases' welche Anwendungsfälle veranschaulichen.

Nach der Implementation erster kleiner Features, entschloss ich mich die ganze Sache zu Refactoren. Dies hat uns die folgenden Arbeiten sehr erleichtert. Die Komponenten waren danach kleiner, unabhängiger und ihre Kompetenzen besser definiert.

Das Entwickeln mit UnitTests war sehr hilfreich. Am Anfang konnte ich dadurch den bestehenden Code etwas kennenlernen, dann hat es beim Refactoring geholfen Fehler aufzudecken und schlussendlich ist es auch bei neu entwickelten Features enorm nützlich. Ist ein Bug aufgetreten habe ich jeweils einen Test erstellt oder bisherige Tests korrigiert, wodurch derselbe Bug in Zukunft leicht erkannt werden konnte.

Das Erstellen einer Campus Karte für unseren Showcase, mit ähnlichem Funktionsumfang wie der interaktive Lageplan, ging mit Khtmlib erstaunlich schnell. Dies im Vergleich mit den Arbeiten am interaktiven Lageplan. Das während dem Projekt gewonnene Wissen hatte sich hierbei ausgezahlt.

A.2.3 Projektarbeit

Das Arbeiten im 2er Team mit Stefan Kemper hat gut funktioniert. Wir haben oft getrennt Entwickelt. Dies vorallem weil wir durch unsere Teilzeit-Jobs an unterschiedlichen Tagen gebunden waren. Bei der Koordination war dabei unser Online Task Tracker im Redmine ein wichtiger Punkt. Durch die konsequente Erfassung von Tasks, Bugs und die Möglichkeit Tasks sich selbst oder dem Partner zuweisen zu können wurde uns die Arbeit sehr erleichtert.

Die Zusammenarbeit mit Herr Zwischenbrugger war für uns sehr Wertvoll. Als Maintainer des Khtmlib Projektes konnte er uns wichtige Tipps geben. Bei der Verständigung war es vorallem Wichtig zu sehen, dass wir vom selben redeten. Am besten ging der Austausch über konkrete Codebeispiele.

Während den wöchentlichen Sitzungen konnten wir jeweils auf das Wissen von Herr Keller im GIS Umfeld zurückgreifen. Dadurch mussten wir nicht alles selbst Erarbeiten und konnten erarbeitetes Wissen kontrollieren.

Ich wünsche dem Khtmlib Projekt das beste für die Zukunft. Gegenüber der Konkurrenz wird es standhalten können wenn es in seiner Nische — Smartphones und Tables — die Nase vorne behält und das ganze Niveau (Technisch, Dokumentation, Be-

kanntheit, usw.) verbessern oder zumindest halten kann. Ich bin mir sicher Herr Zwischenbrugger werden die Ideen und die Lust am Weiterentwickeln nicht ausgehen.

B GLOSSAR

B.1 CANVAS

Ist ein HTML-Element aus dem HTML5-Standard, welches ein dynamisches rendern von Bitmap-Grafiken erlaubt.[1]

B.2 FEATURE / FEATURE TYPE

Eine Feature Instance ist beispielsweise ein konkreter Fluss. Die Feature Instance ist damit vom Feature Type „Fluss“.[2]

B.3 EPSG

EPSG steht für European Petroleum Survey Group. Die EPSG war eine Arbeitsgruppe von europäischen Öl- und Gaserkundungsfirmen. Diese Arbeitsgruppe führte ein System von weltweit einzigartigen Schlüsseln für Koordinatensysteme ein welches noch heute für die Identifikation verschiedener Koordinatensysteme verwendet wird.[3]

B.4 GEOJSON

GeoJSON basiert auf JSON (Technologie zum übermitteln von Daten... eingesetzt bei Ajax oder JOD) und wird verwendet um GeoDaten zu transferieren/nachzuladen. GeoJSON wird unter anderem von OpenLayers eingesetzt und kann auch in Mapnik verwendet werden.[4]

1 Quelle: http://de.wikipedia.org/wiki/Canvas_%28HTML-Element%29
2 Quelle: http://de.wikipedia.org/wiki/Web_Feature_Service
3 Quelle: http://de.wikipedia.org/wiki/European_Petroleum_Survey_Group_Geodesy
4 Quelle: http://en.wikipedia.org/wiki/GeoJSON

B.5 GIS

Abkürzung für Geoinformationssystem

B.6 GML

Abkürzung für Geography Markup Language. GML wird für den Austausch Raumbezogener Daten/Objekte verwendet (auch „Features“ genannt). [5] [6]

KML steht für Keyhole Markup Language und wird über Google weltweit verbreitet. GML ist geeignet, den Inhalt geobezogener Dokumente zu erschließen, indem es ein Spektrum von Anwendungsobjekten und deren Eigenschaften (wie Brücken, Straßen, Bojen und Fahrzeugen) beschreibt. KML dagegen steht für die Visualisierung geographischer Informationen und kann verwendet werden, um GML-Inhalte darzustellen. Andererseits kann auch GML so erweitert werden, dass es Inhalte in der Art von KML darstellen kann. [7]

B.7 GPS

GPS steht für Global Positioning System und ist ein Satelitengestütztes System zur Bestimmung der Position auf der Erde.[8]

B.8 GPX

GPX steht für GPS Exchange Format und ist ein Dateiformat zum speichern von GPS Daten.[9]

B.9 GWC

GWC steht für GeoWebCache und ist eine Java Web Applikation, welche es erlaubt Geodaten in form von Kacheln zu cachen. [10]

---

5 Details siehe: http://de.wikipedia.org/wiki/Geography_Markup_Language
6 Standard: http://www.opengeospatial.org/standards/gml
7 Quelle: http://de.wikipedia.org/wiki/Geography_Markup_Language#GML_und_KML
8 Quelle: http://de.wikipedia.org/wiki/Global_Positioning_System
9 Quelle: http://de.wikipedia.org/wiki/GPS_Exchange_Format
10 Quelle: http://geowebcache.org

B.10 KML[1]

Keyhole Markup Language (KML) ist eine Auszeichnungssprache zur Beschreibung von Geodaten für die Client-Komponenten der Programme Google Earth und Google Maps. KML befolgt die XML-Syntax. [11] [12]

B.11 OGC

Abkürzung für Open Geospatial Consortium

B.12 OPENLAYERS

OpenLayers ist eine JavaScript-Bibliothek, die es ermöglicht Geodaten im Webbrowser anzuzeigen. Bei OpenLayers handelt sich um eine Programmierschnittstelle, die eine clientseitige Entwicklung unabhängig vom Server zulässt. [13]

B.13 OSM

Abkürzung für Open Street Map

B.14 PANNING

Panning ist der Fachbegriff für das verschieben der Karte mit der Maus oder dem Finger.

B.15 SHP/SHAPEFILE

Ist ein Fileformat für Geodaten. [14]

11 Quelle: http://de.wikipedia.org/wiki/Keyhole_Markup_Language
12 Referenz: http://code.google.com/apis/KML/documentation/KMLreference.html
13 Quelle: http://de.wikipedia.org/wiki/Open_layers
14 Quelle: http://de.wikipedia.org/wiki/Shapefile

B.16 SLD

Steht für Styled Layer Descriptor. Bei SLD handelt es sich um einen OGC-Standard für das Styling von Layers. [15]

B.17 SRS

SRS steht für Spantial Reference System. Es handelt sich dabei um ein Koordinatenreferenzsystem. [16]

B.18 SVG

Abkürzung für Scalable Vector Graphics. Bei SVG-Dateien handelt es sich um Vektorbasierte Bilddaten. [17]

B.19 TILE / KACHEL

Eine Kachel ist ein Ausschnitt der Karte. Diese können im (Browser-)Cache abgelegt und wiederverwendet werden.

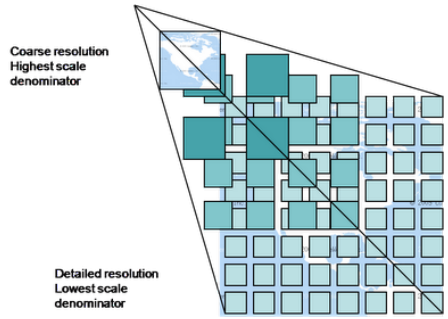


Abbildung B.1: Kacheln (Quelle: http://www.cubewerx.com/products/wmts)

B.20 TMS[11]

TMS steht für Tile Map Service. TMS liefert dem Benutzer vorgerenderte Kachelbilder zurück.

15 Details siehe: http://www.opengeospatial.org/standards/sld
16 Quelle: http://en.wikipedia.org/wiki/Spatial_reference_system
17 Details siehe: http://de.wikipedia.org/wiki/Scalable_Vector_Graphics

B.21 VML

Wird von Microsoft im Internetexplorer zur Darstellung von Vektorgrafiken verwendet. VML (Vector Markup Language) wurde unter anderem von Microsoft entwickelt. Das W3C entwickelte aus VML und PGML den aktuellen Standard SVG. [18]

B.22 WCS

Abkürzung für Web Coverage Service. WCS ist ein Service, der Rasterdaten aufgrund eines ihm übermittelten Requests zurückliefert. [19] [20]

B.23 WFS [9]

WFS steht für Web Feature Service[21] . WFS verwendet GML zur Rückgabe der auf der DB angefragten Geo-Objekten.

B.24 WGS84

WGS84 ist die Bezeichnung für das GPS-Koordinatensystem. Das System teilt den Globus in Längen- und Breitengrade (in englisch longitude und latitude) ein.

B.25 WMS [8]

Ein Web Map Service (WMS) ist eine Schnittstelle zum Abrufen von Auszügen aus Landkarten über das World Wide Web.[22] Die Karten können sowohl aus Rasterdaten (WCS) wie auch aus Vektordaten (WFS) visualisiert werden. Der WMS ist für die Visualisierung der Geo-Daten zuständig.

B.26 WMTS [5]

WMTS steht für Web Map Tiling Service.

18 Quelle: http://de.wikipedia.org/wiki/Vector_Markup_Language
19 Quelle: http://de.wikipedia.org/wiki/Web_Coverage_Service
20 Standard: http://www.opengeospatial.org/standards/wcs
21 Quelle: http://en.wikipedia.org/wiki/Web_Feature_Service
22 Quelle: http://de.wikipedia.org/wiki/Web_Map_Service

B.27 WORLD FILE

Ist ein Zusatzfile zu Bilddateien welches die Georeferenzdaten eines Bildes enthält. Dabei ist der Name der Datei gleich wie das Bild jedoch mit anderer Dateiänderung (beispielsweise .jgw für JPEG oder .pgw für PNG) [23]

B.28 XAPI

XAPI steht bei OSM für Extended API. Diese API ist read-only und bietet erweiterte funktionalitäten für das Durchsuchen von Geodaten. [24]

B.29 YUI

YUI ist eine JavaScript-Bibliothek, welche es erlaubt, auf einfache Art und Weise innerhalb einer Webseite weitere Requests abzusetzen.[25] YUI wird im HSR-Lageplan eingesetzt.

23 Quelle: http://de.wikipedia.org/wiki/World_file
24 Details siehe: http://wiki.openstreetmap.org/wiki/Xapi
25 Details siehe: http://developer.yahoo.com/yui/

LITERATURVERZEICHNIS

[1] Google. *KML Reference*. Number KML 2.2. Google, 2011. URL http://code.google.com/intl/de/apis/kml/documentation/kmlreference.html.

[2] Peter Michaux, 10 2007. URL http://michaux.ca/articles/transitioning-from-java-classes-to-javascript-prototypes.

[3] Dan North, 03 2006. URL http://dannorth.net/introducing-bdd/.

[4] OGC. *OpenGIS KML Encoding Standard (OGC KML)*. Number OGC 07-147r2. OGC, 07. URL http://www.opengeospatial.org/standards/kml/.

[5] OGC. *OpenGIS Web Map Tile Service Implementation Standard*. Number OGC 06-103r4. OGC, 07. URL http://www.opengeospatial.org/standards/wmts.

[6] OGC. *OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture*. Number OGC 06-103r4. OGC, 11 2005. URL http://www.opengeospatial.org/standards/sfa.

[7] OGC. *OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option*. Number OGC 05-134. OGC, 11 2005. URL http://www.opengeospatial.org/standards/sfs.

[8] OGC. *OpenGIS Web Map Service (WMS) Implementation Specification*. Number OGC 06-042. OGC, 2006. URL http://www.opengeospatial.org/standards/wms.

[9] OGC. *OpenGIS Web Feature Service (WFS) Implementation Specification*. Number OGC 09-025r1 (ISO 19142). OGC, 2009. URL http://www.opengeospatial.org/standards/wfs.

[10] OLMCS2011, 02 2011. URL http://www.slideshare.net/cedricmoullet/openlayers-mobile-code-sprint-2011.

[11] OSGeo. *Tile Map Service Specification*. Number TMS 1.0.0. OSGeo, 2011. URL http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification.

[12] Christopher Schmidt. "who needs more than one finger?" — android, 02 2011. URL http://crschmidt.net/blog/archives/455/who-needs-more-than-one-finger-android/.

ERKLÄRUNG

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selbst und ohne fremde Hilfe durchgeführt haben, ausser denjenigen, welche explizit in der Aufgabenstellung erwähnt sind oder mit dem Betreuer schriftlich vereinbart wurden,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.

Rapperswil, Juni 2011

Stefan Kemper

Florian Hengartner