

OpenLayers -

Funktionen, Möglichkeiten und Anwendungen

Diplomarbeit

Als PG nach der SPO 2000

Ausgeführt für die Diplomprüfung am Ende des Wintersemesters 2008 / 2009

(@r01!n W3n93r73r

Carolin Wengerter

geboren am 23. Juli 1983

Matrikelnummer: 415547



HOCHSCHULE FÜR TECHNIK STUTTGART
STUTTGART UNIVERSITY OF APPLIED SCIENCES

Studiengang Vermessung und Geoinformatik

Erstprüfer und Betreuer: Prof. Dr.-Ing. Franz-Josef Behr
Zweitprüfer: Prof. Dr.-Ing. Hardy Lehmkuhler
Betreuer (LVG): Dipl.-Ing. (FH) Manfred Zerndl

Inhaltsverzeichnis

Übersicht.....	5
Abstract.....	5
Abkürzungsverzeichnis.....	6
1 Einleitung.....	8
1.1 Einführung.....	9
1.2 Relevante Standards und Normen.....	9
1.3 Gliederung der Arbeit.....	10
2 OpenLayers.....	10
2.1 Testumgebung.....	11
2.1.1 Firebug.....	11
2.1.2 OpenLayers-Bibliothek.....	13
2.2 Erstes Beispiel.....	13
2.3 Informationsquellen.....	16
3 Steuerelemente von OpenLayers.....	18
3.1 Navigation und PanZoom.....	18
3.2 PanZoomBar.....	19
3.3 MousePosition.....	20
3.4 Permalink.....	20
3.5 Attribution.....	21
3.6 Maßstab.....	21
3.7 LayerSwitcher.....	23
3.7.1 Farben.....	24
3.7.1.1 Hintergrund.....	24
3.7.1.2 Schrift.....	24
3.7.1.3 Deaktivierte Ebenen.....	25
3.7.2 Maximierter LayerSwitcher.....	25
3.7.3 Ausblenden von Ebenen.....	26

3.7.4 Anpassen der Überschriften.....	26
3.8 Weitere Steuerelemente.....	27
4 Integration von standardkonformen Geodiensten.....	27
4.1 WMS.....	27
Performance-Steigerung.....	31
I. Kachelung von Google Maps.....	32
II. Kachelung in OpenLayers.....	33
III. TileCache.....	35
III.I. Visualisierung durch die WMS-Klasse.....	37
III.II. Darstellung durch die TileCache-Klasse.....	37
III.III. Anzeige durch die TMS-Klasse.....	38
III.IV. Performance-Test.....	39
III.V. Gitterberechnung.....	39
IV. Direkter Zugriff.....	41
V. WMS-Zugriff mit mehreren URLs.....	43
4.2 WFS.....	44
4.2.1 Darstellung von Punktobjekten.....	45
4.2.2 Darstellung von Polygonen.....	48
4.3 GeoRSS.....	51
4.3.1 Allgemein.....	51
4.3.2 Visualisierung von georeferenzierten Objekten.....	52
4.3.2.1 Visualisierung durch die Klasse GeoRSS.....	53
4.3.2.2 Visualisierung durch die Klasse GML.....	53
4.3.3 Mögliche Anwendungsgebiete.....	55
5 Integration von proprietären Geodaten.....	56
5.1 Google Maps.....	58
5.2 Yahoo! Maps	60
5.3 Live Search Maps.....	61
5.4 Map24.....	62
5.5 OpenStreetMap.....	64

6 Erfassen, Editieren und Messen.....	66
Beispiel: GeoEditor.....	68
6.1 SVG.....	68
6.2 Ausgabe der Geometrie als WKT.....	69
6.3 Messen.....	70
7 Passwortschutz.....	72
7.1 Passwort in der URL.....	73
7.2 Passwort im HTTP-Kopf.....	73
7.3 Passwort im Proxyskript.....	74
8 Schluss.....	77
Anhang.....	81
Anhang A: UML-Diagramm.....	81
Anhang B: WMS, TileCache, mehrere URLs, Messen.....	82
Anhang C: Proprietäre Geodaten.....	85
Anhang D: Map24.....	88
Anhang E: Direkter Zugriff.....	89
Anhang F: Proxyskript.....	91
Anhang G: WFS-Punktobjekte.....	93
Anhang H: WFS-Polygone.....	95
Anhang I: GeoRSS.....	97
Anhang J: Erfassen und Editieren.....	98
Anhang K: Passwortschutz.....	99
Glossar.....	100
Literaturverzeichnis.....	104
Erklärung.....	106

Übersicht

OpenLayers ist eine quelloffene Bibliothek zur Darstellung von dynamischen Karten in Webbrowsern.

Diese Diplomarbeit untersucht die vielversprechenden Funktionen von *OpenLayers*. Sie stellt die vielfältigen Möglichkeiten für den Einsatz von *OpenLayers* dar und geht dabei besonders auf die Nutzung von *Web Map Services* ein. In diesem Zusammenhang kommen Kachelung, Zwischenspeicherung und direkter Datenzugriff zur Performance-Steigerung zur Sprache, sowie der Zugriff auf passwortgeschützte Daten.

Das Thema der Diplomarbeit wurde vom *Bayerischen Landesamt für Vermessung und Geoinformation* (LVG) in München gestellt und dort bearbeitet.

Abstract

OpenLayers is an open source library for displaying dynamic maps in the internet. *OpenLayers* is a free *application programming interface* (API) and is executed on the client side.

This diploma thesis explores the promising functions of *OpenLayers*. It demonstrates the multifaceted opportunities for the use of *OpenLayers*. Especially the usage of *web map services* will be described. In this context, tiling, caching and direct access for performance acceleration will be discussed. Additionally, the access to password-protected data will be covered.

The subject of the diploma thesis was given by the *Bayerisches Landesamt für Vermessung und Geoinformation* (LVG, Bavarian state office for surveying and geoinformation) in Munich. There the work was executed.

Schlagwörter / keywords:

OpenLayers, WMS, TileCache, WFS, JavaScript, Web-Map-Browser, SVG

Abkürzungsverzeichnis

Ajax	Asynchronous JavaScript and XML
Amazon S3	Amazon Simple Storage Service
API	Application Programming Interface
BKG	Bundesamt für Kartographie und Geodäsie
BSD	Berkeley Software Distribution
BVV	Bayerische Vermessungsverwaltung
CC	Creative Commons
CGI	Common Gateway Interface
DOM	Document Object Model
dpi	dots per inch
ECMA	European Computer Manufacturers Association
EPSG	European Petroleum Surveying Group
FAQ	Frequently Asked Questions
GIS	Geoinformationssystem
GML	Geography Markup Language
GPS	Global Positioning System
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifikationsbezeichnung
ISO	International Organization for Standardization
LVG	Bayerisches Landesamt für Vermessung und Geoinformation
OGC	Open Geospatial Consortium
OSGeo	Open Source Geospatial Foundation
OSM	OpenStreetMap
RFC	Request for Comments
RSS	Really Simple Syndication
SAPOS	Satellitenpositionierungsdienst
SRS	Spatial Reference System

SVG	Scaleable Vector Graphics
TK	Topographische Karte
TMS	Tile Map Service
UK	Übersichtskarte
UML	Unified Modeling Language
UMN	University of Minnesota
URL	Uniform Resource Locator
VML	Vector Markup Language
W3C	World Wide Web Consortium
WFS	Web Feature Service
WFS-T	Transactional Web Feature Service
WGS	World Geodetic System
WKT	Well Known Text
WMS	Web Map Service
WMS-C	Web Map Service - Cached
XML	Extensible Markup Language

1 Einleitung

Das *Open Geospatial Consortium* (OGC) definierte im April 2000 die sogenannte *Web Map Server Implementation Specification* (WMS). Dadurch wurde ein Standard festgelegt, mit dem man über eine URL einen beliebigen Kartenausschnitt eines oder mehrerer Kartenwerke (*Layers*) in einer bestimmten Auflösung anfordern kann. Dieser Standard verbreitete sich aufgrund seiner relativ einfachen Systematik rasch und fand breite Unterstützung bei den meisten Geoinformationssystemen und diversen Kartenviewern im Internet.

Ein WMS bestimmt aus dem zugrunde liegenden Datenbestand die für die Anfrage benötigten Daten. Diese werden intern zu einer großen Datei gerendert. Der geforderte Ausschnitt wird daraus ausgeschnitten und gegebenenfalls skaliert. Diese Rasterdatenoperationen (Kleben, Schneiden und Skalieren) stellen hohe Anforderungen an die Rechnerressourcen bezüglich Rechenleistung und Speicherbedarf und treten bei jedem Aufruf erneut auf. Viele gleichzeitige WMS-Anfragen verursachen auf dem Server eine hohe Auslastung.

Als im Jahre 2005 die Firma *Google* den Kartenviewer *Google Maps* im Internet zur Verfügung stellte, wurde der Bereich der Internet-Kartenviewer revolutioniert. In *Google Maps* wird nicht mit Hilfe eines WMS eine große Datei angefordert und dargestellt, sondern es wurde unter anderem der Datenzugriff optimiert, sodass viele kleine Kacheln asynchron angefordert und im Browser des Benutzers aneinandergesetzt werden. Durch die im Voraus geladene Daten kann der Kartenausschnitt lückenlos verschoben werden. Diese Kacheln müssen nicht bei jedem Zugriff berechnet werden, sondern werden bereits vorge-rechnet auf dem Server in einer bestimmten Systematik abgelegt.

Mit der Verfügbarkeit des Kartenviewers *OpenLayers* der *Open Source Community* ist es sehr leicht möglich, einen eigenen Kartenviewer zu erstellen, der unter anderem diese Art des Kartenzugriffs einsetzt. Im Rahmen dieser Diplomarbeit ist zu prüfen, inwieweit das *OpenLayers*-Framework¹ eingesetzt werden kann und was zu tun ist, um die aufkommende Serverlast zu vermindern. Daneben werden einige weitere Möglichkeiten von *OpenLayers* beleuchtet.

1 Programmiergerüst, das Funktionen und Klassen für den Aufbau einer Anwendung enthält, s. Glossar

Da *OpenLayers* eine noch relativ neue Technologie ist, gibt es über diese keine Fachliteratur. Die Dokumentation lässt sehr zu wünschen übrig und ist überwiegend in englischer Sprache verfasst. Auch unter diesen Gesichtspunkten betrachtet, war es notwendig sich in die Thematik einzuarbeiten und die diversen Möglichkeiten von *OpenLayers* zu untersuchen.

1.1 Einführung

Eine kurze Einführung in das Themengebiet der Internettechnologie soll den Einstieg erleichtern.

Die Umsetzung einer Internetseite wird nach Struktur und Formatierung getrennt. Die Auszeichnungssprache HTML beschreibt die Struktur und *Cascading Stylesheets* (CSS) stellen Formatvorlagen dar. *JavaScript*² ergänzt HTML und macht dynamische und funktionale Internetseiten möglich. *OpenLayers* ist eine *JavaScript*-Bibliothek, die es möglich macht, einen Kartenviewer in eine Internetseite zu platzieren.

1.2 Relevante Standards und Normen

Folgende Normen und Standards sind für die vorliegende Arbeit relevant:

- ISO³-Normen: GML und UML
- W3C⁴-Standards: HTTP, HTML, XML, GML, DOM, CGI, SVG und RSS
- OGC⁵-Standards: WMS, WFS, GML und WKT
- ECMA-Standard 262⁶: Sprachkern von *JavaScript*
- RFC-Quasistandard 3986⁷: URL

2 Skriptsprache, die zur Laufzeit vom Webbrowser interpretiert wird. S. Glossar

3 www.iso.org/iso/home.htm [3.12.08]

4 Das World Wide Web Consortium schafft technische Standards im Internet, s. Glossar, www.w3.org [3.12.08]

5 Das Open Geospatial Consortium definiert Standards im Geoinformationsbereich, s. Glossar, www.opengeospatial.org [3.12.08]

6 www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

7 tools.ietf.org/html/rfc3986 [11.12.08]

Sie werden im Verlauf des Textes oder im Glossar erklärt.

1.3 Gliederung der Arbeit

Im zweiten Kapitel werden *OpenLayers* und der Aufbau der Arbeitsumgebung vorgestellt sowie ein kleines Codebeispiel erklärt. Abschließend werden die Hilfemöglichkeiten aufgezeigt.

Das Framework *OpenLayers* bietet viele Steuerelemente, mit deren Hilfe man die Funktionalität der Kartenanwendung anpassen kann. Einen Überblick über die Steuerelemente verschafft das Kapitel 3.

Schließlich wird im vierten Kapitel auf die Nutzung des *Web Map Service* eingegangen. In diesem Zusammenhang werden die Möglichkeiten zur Performance-Steigerung erörtert. Außerdem werden ein *Web Feature Service* und *GeoRSS-Feed* in *OpenLayers* eingebunden.

Das fünfte Kapitel beschreibt die Möglichkeiten von *OpenLayers* diverse proprietäre Kartenwerke wie *Google Maps* und *OpenStreetMap* einzubinden.

Zusätzlich enthalten die folgenden Kapitel Informationen zur Einbindung von Digitalisier- und Editierfunktionen (Kapitel 6). Der letzte Teil (7. Kapitel) befasst sich mit dem Zugang zu einem passwortgeschützten WMS.

2 OpenLayers

Im Kern befasst sich die vorliegende Arbeit mit *OpenLayers*.

OpenLayers ist eine *JavaScript*-Bibliothek für das Erstellen eines Viewers und somit zum Anzeigen von dynamischen Karten in Webbrowsern. Sie ist mit den *Application Programming Interfaces* (API)⁸ von *Google Maps* oder *Live Search Maps* vergleichbar, allerdings können serverunabhängig Geodaten jeglicher Quelle mit *OpenLayers* dargestellt werden. Die OGC-Standards WMS und WFS werden ebenfalls von *OpenLayers* unter-

⁸ Schnittstelle für Anwendungsprogrammierung, s. Glossar

stützt. *OpenLayers* ist als Open Source Framework mit Hilfe von *Prototype.js*⁹ und *Rico*¹⁰ objektorientiert geschrieben und unter der BSD-Lizenz¹¹ frei verfügbar. Damit werden dem Programmierer über sogenannte *Klassen* bestimmte Funktionalitäten zur Verfügung gestellt. So gibt es beispielsweise die Klasse *WMS*, die den Zugriff auf einen bestimmten *Web Map Service* als Kartenebene (*Layer*) realisiert. *OpenLayers* wurde von der Firma *MetaCarta* entwickelt und am 26. Juni 2006 erstmals der Öffentlichkeit zugänglich gemacht. Jetzt entwickelt die Open-Source-Software-Gemeinschaft das Framework weiter (vgl. OL 2008). Seit 29. September 2008 ist auf der offiziellen Homepage von *OpenLayers*¹² die Version 2.7 erhältlich. *OpenLayers* ist ein Projekt der *Open Source Geospatial Foundation* (OSGeo)¹³. Das geografische Wiki¹⁴ *OpenStreetMap*¹⁵ nutzt ebenfalls *OpenLayers* zur Darstellung der Karten.

2.1 Testumgebung

Nach der Vorstellung von *OpenLayers* beschreibt der folgende Abschnitt die Arbeitsumgebung.

Auf einem PC mit dem Betriebssystem SUSE *Linux* in der Version 10.2 und der grafischen Benutzeroberfläche KDE 3.5 wird das Projekt entwickelt. Als Webserver wird die freie Software *Apache* Version 2.2 eingesetzt. Der Webbrowser *Firefox* mit der Erweiterung *Firebug* werden verwendet.

2.1.1 Firebug

Der *Firebug* ist ein vielseitiges Werkzeug für die Entwicklung von Webseiten. *Firebug* ist ein sogenanntes *Add-on* für den *Firefox*. Man kann ihn kostenlos von der Homepage <http://getfirebug.com> herunterladen. Nach der Installation befindet sich in der rechten

9 Freies JavaScript-Framework, das diverse Programmierhilfen enthält. S. Glossar

10 Open-source JavaScript-Framework, das diverse Funktionen beinhaltet. S. Glossar

11 Die Software darf frei verwendet und ohne Quelltextveröffentlichung verbreitet werden. S. Glossar

12 www.openlayers.org [25.11.08]

13 Organisation zur Förderung freier und quelloffener Software im Geoinformatikbereich. S. Glossar

14 Kollaborative Wissensdatenbank, in der Benutzer Inhalte lesen und ändern können. S. Glossar

15 www.openstreetmap.org [25.11.08]

unteren Ecke des *Firefox*-Fensters ein kleines Icon, das einen Käfer zeigt. Der Käfer ist das Logo des *Firebugs* (s. Abb. 1). Mit einem Klick auf das Icon öffnet sich das Programm *Firebug*. Man kann es auch mit der F12-Taste starten. Der *Firebug* kann im unteren Fensterbereich des Webbrowsers oder in einem eigenen Fenster betrieben werden. Die Oberfläche

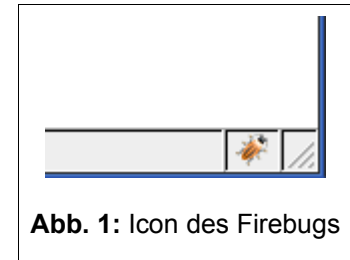


Abb. 1: Icon des Firebugs

von *Firebug* zeigt die Abbildung 2. Im oberen Fensterbereich des *Firebugs* befinden sich zwei Reihen von Schaltflächen, in der Abbildung grau dargestellt. Die untere Reihe bleibt unverändert, die Obere ändert sich in Abhängigkeit von der gedrückten Schaltfläche der unteren Leiste. Die untere Leiste beinhaltet folgende Schaltflächen: *Konsole*, *HTML*, *CSS*, *Skript*, *DOM* und *Netzwerk*. In der *Konsole* kann man über die Eingabezeile Befehle absetzen, die direkt im Webbrowser ausgeführt werden. In der Abbildung 2 sieht man zum Beispiel den schon ausgeführten Befehl `map.getScale()`. Damit wurde der aktuelle Maßstab abgefragt. Die Antwort ist darunter zu sehen. Wenn im Webbrowser ein Fehler auftritt, so wird er im rechten unteren Bereich des Fensters angezeigt, dort wo normalerweise das Logo des *Firebugs* zu sehen ist. Durch einen Klick auf die Fehlermeldung öffnet sich der *Firebug* und in der Konsole werden weitere Informationen zu dem aufgetretenen Fehler ausgegeben.

Über die Schaltfläche *HTML* kann man sich den Inhalt der Internetseite anzeigen lassen. Man kann sogar die Inhalte live bearbeiten. Das ist zum Beispiel für *JavaScript*-Seiten sehr hilfreich, da damit zahlreiche Elemente erst zur Laufzeit erzeugt werden. Mit der Schaltfläche *Untersuchen* in der oberen Zeile ist es möglich,

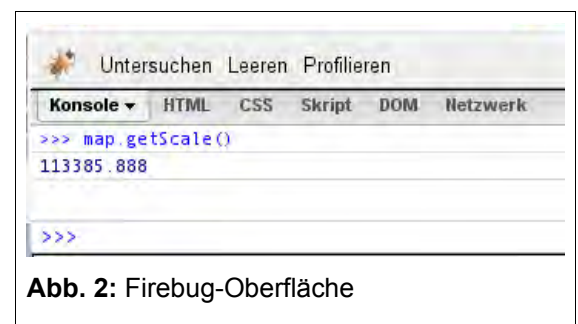


Abb. 2: Firebug-Oberfläche

Bestandteile der Internetseite mit der Maus auszuwählen. Der zugehörige Code wird anschließend im *Firebug* angezeigt. Der Schalter *CSS* zeigt die verwendeten *Cascading Stylesheets*.

Mit dem Knopf *Skript* ist es möglich, Haltepunkte in *JavaScript*-Dateien zu setzen. Beim Laden der Internetseite wird dann in diesem Punkt gestoppt und es kann Schritt für Schritt oder Haltepunkt für Haltepunkt das Programm durchlaufen werden. Währenddessen kann man Werte von Variablen abfragen.

DOM schaltet auf die Ansicht des *Document Object Models* um.

Die schließlich letzte Schaltfläche *Netzwerk* stellt alle ein- und ausgehenden Anfragen dar. Zu jeder Anfrage werden Informationen über die URL, den Statuscode, die Zieldomäne, die Dateigröße und die Dauer des Ladevorgangs angezeigt. Außerdem können weitere Details zum Anfrage- und Antwort-Kopf der Anfrage ausgegeben werden. Durch einen Rechtsklick auf eine Anfrage kann man die Anfrage in einem eigenen Fenster öffnen. Dies ist zum Beispiel sinnvoll, um weitere Informationen wie den Fehlercode zu erhalten. Nach Beendigung des Ladevorgangs wird die Summe der Anfragen, der Dateigröße und der Zeit ausgegeben. Der Button *Leeren* löscht die Liste der Anfragen.

2.1.2 OpenLayers-Bibliothek

Auf der Homepage von *OpenLayers* befindet sich die neueste Version in zwei verschiedenen Ausführungen. Hinter dem *Link to the hosted version* steckt eine kompakte Datei, die das komplette *OpenLayers*-Framework beinhaltet. Diese hat den Vorteil, dass sie schnell vom Browser geladen werden kann, Dauer ca. 2 - 3 s. Da sie aber keinerlei Formatierungszeichen enthält, ist sie sehr unübersichtlich und für das Auffinden von Fehlern und die Entwicklung ungeeignet. Für diese Aufgaben eignet sich das Paket von 190 formatierten Dateien, das pro Klasse eine Datei enthält. Die Ladezeit beträgt ungefähr 60 s. Ist die Entwicklung abgeschlossen, lassen sich die Dateien mit dem Skript *build.py* komprimieren¹⁶.

2.2 Erstes Beispiel

Nachdem die Arbeitsumgebung für den Einsatz von *OpenLayers* eingerichtet wurde (s. Abschnitt 2.1), kann man mit dem Erstellen eines Kartenviewers beginnen. Als Einstieg eignet sich ein kleines Beispiel¹⁷:

```
<html>
<head>
  <title>OpenLayers Beispiel</title>
  <script type="text/javascript" src="js/OpenLayers27.js"></script>
</head>
<body>
  <div style="width:100%; height:100%" id="map"></div>
  <script type="text/javascript">
```

¹⁶ <http://trac.openlayers.org/wiki/Profiles> [25.11.08]



¹⁷ <http://openlayers.org/doc> [25.11.08]

```
var map = new OpenLayers.Map('map', {maxResolution: 'auto'});
var wms = new OpenLayers.Layer.WMS( "OpenLayers WMS",
    "http://labs.metacarta.com/wms/vmap0", {layers: 'basic'} );
map.addLayer(wms);
map.zoomToMaxExtent();
</script>
</body>
</html>
```

Dieser kurze Code erzeugt bereits einen funktionsfähigen Kartenviewer, wie er in der Abbildung 3 zu sehen ist.



Abbildung 3: Erster Kartenviewer

In dem Viewer kann man mit gedrückter linker Maustaste das Kartenbild verschieben. Die Pfeiltasten navigieren ebenfalls die Karte. Ein Doppelklick vergrößert die Landkarte, genauso wie ein Klick auf die Plustaste . Mit der Minustaste  zoomt man heraus und die Schaltfläche dazwischen, die eine Weltkugel zeigt, schaltet auf das komplette Kartenbild um.

Kern jeder *OpenLayers*-Anwendung ist ein Objekt der Klasse *Map*. Dieses Objekt enthält Ebenen (*Layer*) mit den eigentlichen Daten. In dem *Map*-Objekt werden generelle Karteneinstellungen zur Darstellung gespeichert, wie zum Beispiel Projektion, Zoomstufen und Einheit. Eine Ebene kann beispielsweise ein Kartenwerk beinhalten. *OpenLayers* bietet viele verschiedene Arten von Ebenenklassen an, Details dazu befinden sich in den Kapiteln 4 und 5 zur Datenintegration. Über die Art der *Layer*-Klasse kann *OpenLayers* feststellen, wie es auf die Geodaten zugreifen kann.

Die Umsetzung im Sourcecode beschreibt der nächste Absatz.

Zuerst bindet man die *OpenLayers*-Bibliothek ein. Dafür gibt es zwei Alternativen. Entweder man bezieht die Bibliothek lokal, vorausgesetzt sie wurde zuvor dort gespeichert, oder direkt von der Homepage:

```
<html>
<head>
  <title>OpenLayers Beispiel</title>
  <script type="text/javascript" src="js/OpenLayers27.js"></script>
  // alternativ:
  <script src = "http://openlayers.org/api/OpenLayers.js"></script>
</head>
```

An der Stelle der HTML-Seite, wo die Karte platziert werden soll, muss lediglich ein *div*-Element eingefügt werden:

```
<body>
  <div style="width:100%; height:100%" id="map"></div>
```

Nun beginnt der *JavaScript*-Bereich:

```
<script type="text/javascript">
```

Der *OpenLayers.Map*-Konstruktor generiert das Kartenobjekt. Dazu benötigt man als Argument ein HTML-Element oder die ID eines solchen Elementes ('map'). In dieses Element wird die Karte platziert:

```
var map = new OpenLayers.Map('map', {maxResolution: 'auto'});
```

Im nächsten Schritt wird eine Ebene der Karte hinzugefügt. In diesem Fall wird ein *WMS*-Layer in den Viewer eingebunden. Eine *WMS*-Ebene kann durch Angabe eines Namens, der URL zum *WMS*-Server und einer *WMS*-spezifischen Parameterliste erzeugt werden:

```
var wms = new OpenLayers.Layer.WMS( "OpenLayers WMS",
  "http://labs.metacarta.com/wms/vmap0", {layers: 'basic'} );
```

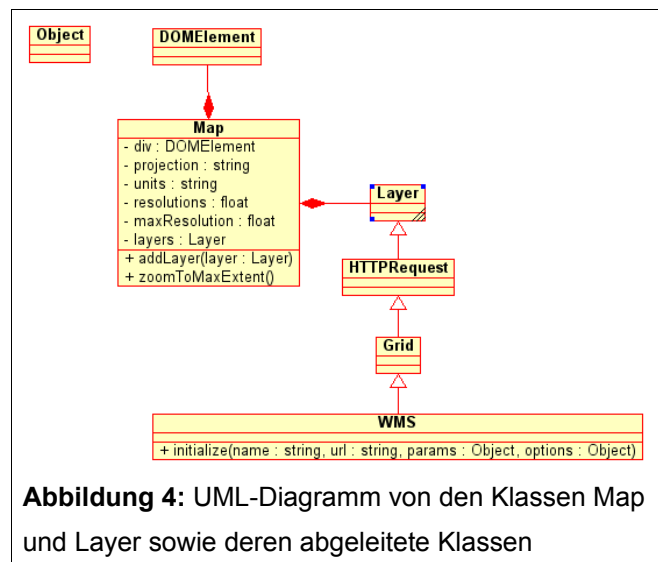
Über die *addLayer*-Funktion des Kartenobjektes *map* wird die Ebene angefügt:

```
map.addLayer(wms);
```

Schließlich muss man den Mittelpunkt und die Zoomstufe festlegen, um die Karte anzuzeigen. Alternativ kann man mit der *zoomToMaxExtent*-Funktion auf den maximal verfügbaren Kartenausschnitt zoomen:

```
map.zoomToMaxExtent();
</script>
</body>
</html>
```

Einen Überblick über die verwendeten Klassen im ersten Beispiel verschafft das UML-Diagramm in Abbildung 4. Dort sind die beiden Klassen *Map* und *Layer* dargestellt. Außerdem sind die in diesem Abschnitt verwendeten Attribute und Operationen sowie die Klasse des *Web Map Service* abgebildet.



In der Abbildung lassen sich die Zusammenhänge erkennen. Die Klasse *Map* beinhaltet unter anderem die Einstellungen für Projektion, Einheit, maximale Auflösung und Ebenen. Des Weiteren wird über das Attribut *div* der Platzhalter für die Karte bestimmt. Neben der Klasse *Map* ist die Klasse *Layer* zu sehen. Zwischen ihnen besteht eine Aggregationsbeziehung. Das bedeutet, dass die Klasse *Layer* eine Komponente in der Klasse *Map* ist. Das Attribut *layers* der Klasse *Map* ist vom Typ *Layer*. Über dieses Attribut findet die Verknüpfung der beiden Klassen *Map* und *Layer* statt. Somit werden Objekte der *Layer*-Klasse dem Attribut *layers* der *Map*-Klasse zugeordnet.

Mit diesem ersten Beispiel sollte der Einstieg in *OpenLayers* gelungen sein. Im folgenden Abschnitt werden Hilfestellungen für das vertiefte Programmieren gegeben.

2.3 Informationsquellen

Im Hinblick auf das Verständnis von *OpenLayers* stehen eine Reihe vertiefender und weiterführender Informationen zur Verfügung. Diese werden in diesem Abschnitt vorgestellt.

Es gibt ein UML-Diagramm mit allen Klassen von *OpenLayers*. Durch das Klassendiagramm kann man sich einen Überblick über die Bibliothek verschaffen. Außerdem hilft es Zusammenhänge zu verstehen. Das Diagramm befindet sich im Anhang A. Man kann es auch online von der *OpenLayers*-Homepage beziehen.

→ trac.openlayers.org/wiki/UML [25.11.08]

Die FAQ bieten Antworten auf generelle und häufig gestellte Fragen von Programmierern und Anwendern an die *OpenLayers*-Gemeinschaft.

→ faq.openlayers.org [25.11.08]

Außerdem ist das Wiki der *OpenLayers*-Homepage eine gute Informationsquelle. Dort werden viele Grundlagen leicht verständlich erklärt.

→ trac.openlayers.org/search [25.11.08]

In der Klassendokumentation werden die einzelnen Klassen von *OpenLayers* beschrieben. Es gibt eine Suchfunktion, über die man nach Attributen und Funktionen suchen kann. Die Dokumentation ist gut für den Einstieg, aber leider unvollständig. Es sind nicht alle Klassen mit allen Attributen und Funktionen beschrieben.

→ dev.openlayers.org/releases/OpenLayers-2.7/doc/apidocs/files/OpenLayers-js.html
[25.11.08]

Früher oder später wirft man einen Blick in den Quelltext von *OpenLayers*. Im Gegensatz zu den restlichen Dokumentationen ist der Quellcode vollständig. Für einen Entwickler empfiehlt es sich mit dem Sourcecode zu arbeiten. Ein Anfänger wird sich eher schwer tun und sollte mit den anderen Informationen beginnen.

Der Sourcecode ist folgendermaßen aufgebaut:

In dem Ordner *lib*, wie *library*, befindet sich die komplette Bibliothek von *OpenLayers*. Dort ist auch die zentrale Datei *OpenLayers.js* abgelegt. Diese Datei wird in HTML eingebunden und lädt die restlichen Dateien nach. In der Datei sind alle *JavaScript*-Dateien der *OpenLayers*-Bibliothek vermerkt. Wenn man in dieser Datei die deutsche Sprachdatei "[OpenLayers/Lang/de.js](#)" einträgt, werden bestimmte Bezeichnungen wie *BaseLayer* und *Scale* ins Deutsche übersetzt. In den Ordner *OpenLayers* sind die grundlegenden Klassen

wie zum Beispiel *Map*, *Layer*, *BaseTypes* eingeordnet. Die abgeleiteten Klassen werden jeweils in einem Ordner mit dem Namen der Basisklasse gespeichert. So befinden sich beispielsweise in dem Ordner *Layer* die abgeleiteten Klassen *WMS*, *Google* und *TileCache*.

Die in der Einführung angesprochenen *Cascading Stylesheets* werden zentral in der Datei *style.css* verwaltet. Die CSS-Datei befindet sich in dem Ordner *theme/default*.

→ trac.openlayers.org/browser/trunk/openlayers/lib [25.11.08]

Lösungen zu speziellen Themen und persönliche Hilfe findet man in der *MailingList*. Die *MailingList* ist in drei Foren unterteilt. So gibt es jeweils ein eigenes Forum für Benutzer, Entwickler und *TileCache*. Eigene Einträge können nach der Anmeldung verfasst werden.

→ www.nabble.com/forum/Search.jtp?forum=15906&local=y [25.11.08]

Schließlich soll noch auf die sehr hilfreiche Quelle der zahlreichen Beispiele hingewiesen werden. Von derzeit 117 Realisierungen anderer Entwickler kann man viel lernen und profitieren.

→ www.openlayers.org/dev/examples/example-list.html [25.11.08]

Dies sind zwar einige Möglichkeiten, um Hilfe zu erhalten, dennoch ist es oft nicht einfach, zu einem spezifischen Problem die Lösung zu finden.

3 Steuerelemente von OpenLayers

Das dritte Kapitel befasst sich mit den individuellen Gestaltungsmöglichkeiten einer Kartenanwendung mit *OpenLayers*.

Das Framework *OpenLayers* liefert viele Steuerelemente (*Controls*) mit deren Hilfe man die Bedienung der Karten vereinfachen und das Aussehen des Viewers beeinflussen kann. Ausgewählte Steuerelemente werden nun im Einzelnen beschrieben.

3.1 Navigation und PanZoom

Standardmäßig gehören zu jeder *OpenLayers*-Karte die beiden Steuerelemente *Navigation* und *PanZoom*. Mit dem Steuerelement *Navigation* ist es möglich, die Karte mit der Maus zu navigieren. So kann man mit einem Doppelklick die Karte vergrößern und mit

gedrückter linker Maustaste die Karte verschieben. Der *PanZoom* stellt diese Funktionen zusätzlich über separate Schaltflächen bereit (s. Abb. 5). Mit dem Plus- und Minuszeichen kann man den Kartenausschnitt vergrößern und verkleinern. Die Weltkugel schaltet auf den gesamten Kartenausschnitt um. Mit den Pfeiltasten verschiebt man das Kartenbild in die entsprechende Richtung.

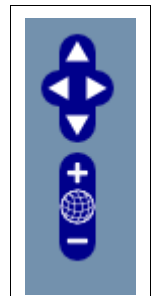


Abb. 5:
PanZoom

Der folgende Code wird von *OpenLayers* intern ausgeführt und muss nicht implementiert werden. Er soll an dieser Stelle der Information dienen.

```
map = new OpenLayers.Map( "map", {controls:[
                                new OpenLayers.Control.PanZoom(),
                                new OpenLayers.Control.Navigation()
                                ]});
```

In den Optionen, also dem zweiten Parameter des *Map*-Objektes, kann man auf das Attribut *controls* verweisen. Der Datentyp des Attributes *controls* ist ein Array von Steuerelementen.

Man kann auch eine statische Karte ohne jegliche Funktionalität generieren. Diese würde sich eventuell für eine Anfahrtsskizze eignen. Dazu fügt man explizit keine Steuerelemente, also einen leeren Array, der Karte hinzu:

```
map = new OpenLayers.Map("map", {controls: []});
```

3.2 PanZoomBar

Die *PanZoomBar*, s. Abb. 6, ist mit dem *PanZoom* vergleichbar, bietet aber bessere Interaktionsmöglichkeiten. Zwischen dem Plus- und Minuszeichen sind hier Schaltflächen für jede Zoomstufe integriert. So ist eine bestimmte Zoomstufe mit einem Klick einzustellen. In den Optionen des Kartenobjektes (*Map*) gibt man hinter dem Attribut *controls* das gewünschte Steuerelement *PanZoomBar* an:

```
controls:[new OpenLayers.Control.PanZoomBar()]
```

Damit wird es dem Kartenviewer hinzugefügt.

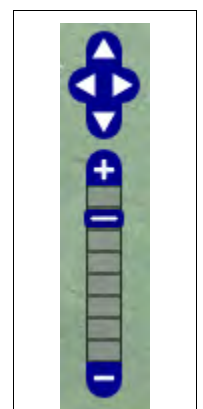


Abb. 6:
PanZoomBar

3.3 MousePosition

Mit dem *MousePosition*-Control werden dem Benutzer ständig die Koordinaten im rechten unteren Bereich der Karte angezeigt (s. Abb. 7). Damit kann sich der Nutzer jederzeit orientieren. Das Steuerelement kann durch seine Angabe unter *controls* der Karte hinzugefügt werden:

```
new OpenLayers.Control.MousePosition()
```

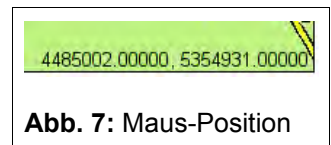


Abb. 7: Maus-Position

3.4 Permalink

Hat der Benutzer einen bestimmten Kartenausschnitt gefunden und will diesen weitersenden oder für sich als Lesezeichen speichern, so benötigt er die URL, den sogenannten *Permalink*, für genau diesen Ausschnitt. Diese URL liefert das Steuerelement *Permalink*. Da der Begriff im deutschsprachigen Raum eher unbekannt ist, sollte man die Beschriftung ändern. Die Übersetzungen der Bezeichnungen sind in einer Sprachdatei eingetragen. Die Sprachdateien befinden sich in dem Ordner *lib/OpenLayers/Lang*. Der Name der deutschen Sprachdatei ist *de.js*. In der Datei kann man den String hinter dem Schlüsselwort *permalink* anpassen:

```
'permalink': "URL zu dieser Seite",
```

Schließlich wird das Steuerelement *Permalink* (s. Abb. 8) auf die gleiche Weise, wie schon bei den anderen Steuerelementen geschehen, der Karte hinzugefügt:

```
new OpenLayers.Control.Permalink()
```

Das Aussehen des Steuerelementes wird im Abschnitt *.olControlPermalink* in der CSS-Datei *style.css* festgelegt. Will man zum Beispiel die Schriftart (*font-family*) Arial festsetzen, so gibt man folgende Zeile an:

```
.olControlPermalink {
  right: 3px;
  bottom: 1.5em;
  display: block;
  position: absolute;
  font-size: smaller;
  font-family: arial;
}
```

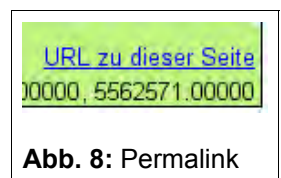


Abb. 8: Permalink

3.5 Attribution

Es gibt zahlreiche Karten, die unter Kopierschutz stehen. Werden urheberrechtlich geschützte Daten in die Anzeige integriert, steht einem das Steuerelement *Attribution* (s. Abb. 9) für die Platzierung des Copyright-Vermerkes zur Verfügung. Das Steuerelement *Attribution* kann man auf alle Ebenen (*Layer*) von *OpenLayers* anwenden. Hier wird es am Beispiel der *WMS*-Ebene erklärt.

Das Steuerelement wird mit der schon bekannten Methode im *controls*-Array eingefügt:

```
new OpenLayers.Control.Attribution()
```

Außerdem muss bei der Ebenendefinition in den Optionen der Anzeigetext festgelegt werden:

```
tk = new OpenLayers.Layer.WMS("TK",
    "http://geodaten.bvvn.de/ogc/ogc_dipl.cgi?",
    {layers: 'tk', format: 'image/png'},
    {attribution: "(c) Bayerische Vermessungsverwaltung"});
```

Wie schon zu Beginn des Kapitels erwähnt, wird die Formatierung über CSS gesteuert. Die entsprechende Datei *style.css* befindet sich im Ordner *theme/default*. In der Datei gibt es einen Eintrag (*.olControlAttribution*) für das *Attributions*-Steuerelement. Dort kann man das Layout nach Belieben anpassen. Ein Beispiel:

```
.olControlAttribution {
    font-size: smaller;
    font-family: arial;
    right: 10px;
    top: 10px;
    position: absolute;
    display: block;
    background-color: white;
    opacity: 0.9;
}
```



Abbildung 9: Copyright-Vermerk

Es wird eine kleine Schriftgröße (*font-size: smaller*) in der Schriftart Arial gewählt und die Position (*right: 10px; top: 10px; position: absolute*) des Copyright-Textes festgesetzt. Zur besseren Lesbarkeit wird die Hintergrundfarbe (*background-color*) auf weiß gesetzt.

3.6 Maßstab

Für die Maßstabsanzeige sind zwei Steuerelemente zuständig, die Maßstabsleiste (*Scale-Line*) und der Maßstab (*Scale*).

Nachdem die Steuerelemente mit folgendem Quellcode hinzugefügt wurden, werden sie in der linken und rechten unteren Ecke im Kartenbild angezeigt (s. Abb. 10):

```
new OpenLayers.Control.ScaleLine(),
new OpenLayers.Control.Scale()
```



Abbildung 10: ScaleLine und Scale

Für die deutsche Darstellung ist die Sprachdatei zuständig. Diese kann durch einen Eintrag in die Datei *OpenLayers.js* zu der *OpenLayers*-Bibliothek hinzugefügt werden.

Die Maßstabsleiste zeigt eine Linie an und deren Länge. Der Maßstab ist das Verhältnis von einer Strecke in der Karte zu der Strecke in der Natur (s. Gleichung 1.1 und 1.2).

$$\text{Maßstab} = 1 : m \quad (\text{Gl. 1.1})$$

$$m = \frac{\text{Naturstrecke}_{[m]}}{\text{Kartenstrecke}_{[m]}} \quad (\text{Gl. 1.2})$$

In der Abbildung 10 entspricht 1 cm in der Karte 113 000 cm in der Natur, also 1,13 km. Dabei muss allerdings folgendes beachtet werden:

Betrachtet man im Browser eine 500 x 500 Pixel große Kachel und misst mit einem Lineal auf dem Bildschirm deren Seitenlänge nach, so erhält man beispielsweise 13,2 cm. Eine Kachel hat in der Natur eine Seitenlänge von 20 000 m, bei einer vorgegebenen Bodenauflösung von 40 m/Pixel (s. Gl. 2).

$$\text{Naturstrecke}_{[m]} = \text{Bildstrecke}_{[\text{Pixel}]} \cdot \text{Bodenauflösung}_{\left[\frac{m}{\text{Pixel}}\right]} \quad (\text{Gl. 2})$$

$$\text{Naturstrecke [m]} = 500 \text{ Pixel} \times 40 \text{ m/Pixel} = 20\,000 \text{ m}$$

Daraus ergibt sich ein Maßstab von 1 : 151 515 (s. Gl. 1.1 und 1.2).

$$\text{Maßstab} = 0,132 \text{ m} : 20\,000 \text{ m} = 1 : 151515$$

Fragt man aber über die *Firebug*-Konsole mit dem Befehl *map.getScale()* den Maßstab ab, so erhält man 1 : 113 000. Wie lässt sich diese Differenz erklären?

Die Seitenlänge der Kachel, die 500 Pixeln entspricht, wird auf dem Monitor mit einer Länge von 13,2 cm abgebildet. Das entspricht einer relativen Auflösung bzw. Punktdichte von 96 dpi¹⁸ (s. Gl. 3).

$$\text{Punktdichte}_{[\text{dpi}]} = \frac{\text{Bildstrecke}_{[\text{Pixel}]} \cdot 2,54 \frac{\text{cm}}{\text{Zoll}}}{\text{Kartenstrecke}_{[\text{cm}]}} \quad (\text{Gl. 3})$$

$$\text{Punktdichte} [\text{dpi}] = (500 \text{ Pixel} \times 2,54 \text{ cm/Zoll}) : 13,2 \text{ cm} = 96,2 \text{ dpi}$$

OpenLayers nimmt aber standardmäßig eine Punktdichte von 72 dpi an. Diese Konstante kann man wieder über die Konsole des *Firebugs* abfragen. Dazu gibt man einfach den Befehl *OpenLayers.DOTS_PER_INCH* ein. Die relative Auflösung unterscheidet sich nämlich von PC zu PC. Sie ist abhängig von der absoluten Auflösung und Größe des Monitors oder Ausgabegerätes wie zum Beispiel des Druckers. 72 dpi werden als Standard- und Durchschnittswert angenommen. Die absolute Auflösung des Monitors ist zwar bestimmbar, aber die Displaygröße nicht. Unter den genannten Umständen ist es ratsamer, auf die Anzeige des Maßstabes zu verzichten und sich ausschließlich der Maßstabsleiste zu bedienen.

3.7 LayerSwitcher

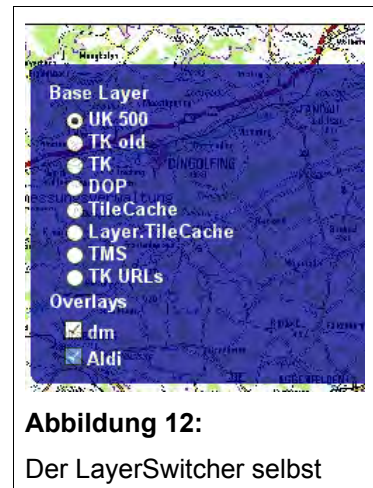
Als vorerst letztes Steuerelement befasst sich der folgende Abschnitt mit dem *LayerSwitcher*. Werden in einem Kartenviewer mehrere Ebenen dargestellt, so kann man zwischen den Ebenen hin- und herschalten. Diese Funktion übernimmt der *LayerSwitcher*. Dabei ist generell zwischen zwei Arten von Ebenen zu unterscheiden. Zum einen gibt es die Grundebenen (*BaseLayer*), von denen jeweils nur eine angezeigt werden kann. Und zum anderen gibt es die Überlagerungsebenen (*Overlays*), die über einer beliebigen Grundebene dargestellt werden können. Die Überlagerungsebenen können jederzeit und unabhängig voneinander über den *LayerSwitcher* ein- und ausgeblendet werden. Das Steuerelement wird mit folgendem Code hinzugefügt:

```
new OpenLayers.Control.LayerSwitcher()
```

Danach erscheint am rechten oberen Rand eine kleine Schaltfläche, die ein weißes Plus auf dunkelblauem Hintergrund zeigt (s. Abb. 11). Mit einem Klick auf diese Schaltfläche öffnet sich der *LayerSwitcher*. Im oberen Bereich sind die Grundebenen dargestellt und

¹⁸ dpi: dots per inch

unten die optionalen Überlagerungen (*Overlays*, s. Abb. 12). Ein Klick auf die Schaltfläche, die nun ein Minuszeichen trägt, schließt den *LayerSwitcher*. In den beiden Abbildungen ist die Standarddarstellung des *LayerSwitchers* zu sehen. Das Layout kann man an seine eigenen Vorstellungen und Wünsche anpassen.



3.7.1 Farben

3.7.1.1 Hintergrund

Die Hintergrundfarbe wird über das Attribut *activeColor* der Klasse *LayerSwitcher* gesteuert. Damit ist es möglich, direkt beim Anlegen des Steuerelementes die Hintergrundfarbe zu ändern:

```
var ls = new OpenLayers.Control.LayerSwitcher({activeColor: 'white'});
```

3.7.1.2 Schrift

Für die Schriftfarbe gibt es leider kein solches Klassenattribut. So kann man die Farbe direkt in der *loadContents*-Funktion der Klasse *LayerSwitcher* ändern:

```
this.div.style.color="black";
```

Will man die Farbe beim Erstellen des *LayerSwitchers* festlegen, so muss zunächst ein neues Klassenattribut angelegt werden. Dieses Attribut erhält den Namen *fontColor* und den Standardwert *weiß*:


```
/**
 * Property: fontColor
 * {String}
 */
fontColor: "white",
```

Außerdem muss in der *loadContents*-Funktion das Attribut *fontColor* zugewiesen werden:

```
this.div.style.color = this.fontColor;
```

Mit diesen beiden Änderungen kann die Schriftfarbe des *LayerSwitchers* direkt beim Erstellen festgelegt werden:

```
new OpenLayers.Control.LayerSwitcher({fontColor: 'black'})
```

Bei solchen Änderungen muss man sich klar darüber werden, dass der Programmcode von *OpenLayers* verändert wurde und diese Änderungen nach einem Versionsupdate erneut vollzogen werden müssen.

3.7.1.3 Deaktivierte Ebenen

Ebenen können maßstabsabhängig aktiviert und deaktiviert werden. Die deaktivierten Ebenen werden in einer anderen Farbe, als die Aktivierten dargestellt. Die Farbe kann man in der *redraw*-Funktion der Klasse *LayerSwitcher* ändern:

```
labelSpan.style.color="gray"
```

Diese Änderungen ergeben den neu gestalteten *LayerSwitcher*, s. Abbildung 13.



Abbildung 13:
Individueller LayerSwitcher

3.7.2 Maximierter LayerSwitcher

Da die Schaltfläche zum Öffnen des standardmäßig zugeklappten *LayerSwitchers* sehr klein und damit leicht zu übersehen ist, ist zu überlegen, das Steuerelement bereits beim Laden der Seite aufzuklappen. Damit erhält der Benutzer gleich einen Überblick über die vorhandenen Ebenen und kann den *LayerSwitcher* bei Bedarf schließen.

Dazu legt man den *LayerSwitcher* bei der Erstellung in einer Variablen ab. Diese fügt man dem Kartenobjekt (*map*) hinzu. Danach kann man das Steuerelement über die Funktion

`maximizeControl()` maximieren:

```
var ls = new OpenLayers.Control.LayerSwitcher;
map.addControl(ls);
ls.maximizeControl();
```

3.7.3 Ausblenden von Ebenen

Wenn ein Kartenviewer nur eine einzige Grundebene enthält, ist es nicht sinnvoll sie im *LayerSwitcher* anzuzeigen, da man sie nicht ausblenden kann. Bei der Definition der Ebene kann man das Attribut `displayInLayerSwitcher` auf `false` setzen, somit wird diese Ebene nicht im *LayerSwitcher* angezeigt:

```
tk = new OpenLayers.Layer.WMS("TK" ,
    "http://geodaten.bvv.bybn.de/ogc/ogc_dipl.cgi?",
    {layers: 'tk', format: 'image/png', srs: 'EPSG:31468', version: '1.1.1'},
    {displayInLayerSwitcher: false});
```

Alternativ kann man auch nach der Ebenendefinition das Attribut ändern:

```
tk.displayInLayerSwitcher = false;
```

Den *LayerSwitcher* ohne Darstellung der Grundebene zeigt die Abbildung 14.

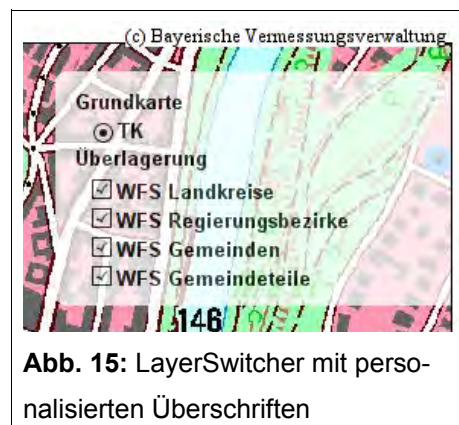


3.7.4 Anpassen der Überschriften

Ist die Zielgruppe des Kartenviewers überwiegend deutschsprachig, so können die Überschriften des *LayerSwitchers* übersetzt werden. Die Änderungen müssen in der deutschen Sprachdatei `de.js` vollzogen werden. In der Datei wird der Wortlaut der Überschriften festgelegt. Dazu gibt man nach dem Schlüsselwort die deutsche Bezeichnung an:

```
'overlays': "&Uuml;berlagerung",
'baseLayer': "Grundkarte",
```

Dabei ist zu beachten, dass Sonderzeichen durch HTML-kodierte Zeichen (*Entities*) um-



geschrieben werden müssen. So wird das \dot{U} mit *Ü* umschrieben. Der *LayerSwitcher* mit den übersetzten Überschriften ist in Abbildung 15 zu sehen.

3.8 Weitere Steuerelemente

Darüber hinaus gibt es noch weitere umfangreiche Steuerelemente, die im Laufe der Arbeit vorgestellt und in *OpenLayers* eingebunden werden. Das Steuerelement *Select-Feature* wird im Abschnitt 4.2.1 behandelt. Ebenso werden behandelt *Panel* und *Toggle*, siehe Abschnitt 4.2.2. Im 6. Kapitel werden die Steuerelemente *EditingToolbar* und *Drag-Feature* vorgestellt. Außerdem wird im Abschnitt 6.3 die Einbindung des Steuerelementes *Measure* beschrieben.

4 Integration von standardkonformen Geodiensten

OpenLayers wurde unter der Vorgabe entwickelt diverse Datenquellen in einem Viewer zu vereinen. In den beiden folgenden Kapiteln wird die Vielfalt an Integrationsmöglichkeiten aufgezeigt. In diesem Kapitel wird auf die standardkonformen Online-Kartendienste eingegangen, wie sie beispielsweise das LVG anbietet.

4.1 WMS

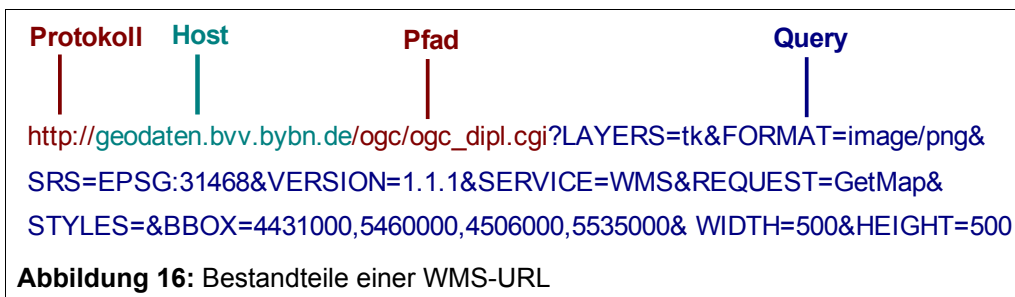
Bereits im Abschnitt 2.2 *Erstes Beispiel* wurde eine WMS-Ebene in *OpenLayers* eingebunden. In diesem Kapitel wird die WMS-Nutzung vertieft (Quelltext s. Anhang B).

Der *Web Map Service* (WMS, engl. Online-Kartendienst) ist eine Spezifikation des OGC, mit dessen Hilfe Geodaten über das Internet ausgetauscht werden können. Dabei müssen Client (Webbrowser) und Server den OGC-Standard unterstützen. Die Karten werden in Form von Rasterbildern geliefert und die Sachdaten im XML¹⁹- oder Textformat.

Beim WMS gibt es vier verschiedene Operationen: *GetCapabilities*, *GetMap*, *GetFeature-Info* und *GetLegendGraphic* (vgl. OGC 2002). Eine WMS-Anfrage wird als URL (*Uniform Resource Locator*, Internetadresse) zum Server geschickt. Die URL setzt sich aus dem Protokoll, dem Host, dem Pfad und dem Querystring zusammen. Die Anfragen werden über

¹⁹ W3C-Standard zur hierarchischen Beschreibung von Daten, s. Glossar

HTTP²⁰ vom Benutzer an den Server gesendet. Hinter dem Protokoll folgt die Angabe der Serveradresse und der Dateipfad. Mit dem Fragezeichen wird die Suchanfrage (*Query*) eingeleitet. Die einzelnen Parameter werden mit dem Und-Zeichen (&) verbunden. Bei dem Beispiel in Abbildung 16 handelt es sich um einen *GetMap*-Aufruf, der, wie der Name schon sagt, Karten bzw. georeferenzierte Rasterbilder liefert.



Mit Angabe der Parameter lässt sich die Anfrage konkretisieren. Die folgenden Parameter sind erforderlich: VERSION, REQUEST, LAYERS, STYLES, SRS, BBOX, WIDTH, HEIGHT und FORMAT. Die beiden Parameter TRANSPARENT und BGCOLOR sind optional. Unter VERSION wird die Version des WMS-Servers angeführt. Nach REQUEST folgt der Anfragetyp wie zum Beispiel *GetMap* oder *GetCapabilities*. Ebenen kann man unter LAYERS angeben und Zeichenvorschriften unter STYLES. Der Parameter SRS bestimmt das räumliche Bezugssystem, das in der Regel mit EPSG²¹-Codes bezeichnet wird. Hinter BBOX steckt die Boundingbox mit der linken unteren und der rechten oberen Ecke des Kartenausschnittes. Die Werte werden in der Reihenfolge *minx*, *miny*, *maxx* und *maxy* angeordnet. Die Breite und Höhe des Kartenfensters kann man aus WIDTH und HEIGHT auslesen. Das gewünschte Datenformat, zum Beispiel *png* oder *jpg*, setzt man mit FORMAT fest. Die beiden optionalen Parameter TRANSPARENT und BGCOLOR legen schließlich die Hintergrundtransparenz und -farbe fest.

Nun wird ein WMS des LVGs in *OpenLayers* eingebunden. Unter den generellen Karteneinstellungen legt man die Ausmaße Bayerns, die Einheit und Projektion fest. In diesem Fall kommt die vierte Zone der Gauß-Krüger-Projektion²² zum Einsatz, die unter dem EPSG-Code 31468 eingetragen ist:

²⁰ Protokoll zur Datenübertragung in einem Netzwerk, s. Glossar

²¹ Weltweit eindeutige Schlüsselnummer für Koordinatensysteme, s. Glossar

²² Deutsches Hauptdreiecksnetz, Bessel-Ellipsoid, Fundamentalpunkt in Potsdam, 10,5 – 13,5 °

```
var options =
{
    maxExtent: new OpenLayers.Bounds(4281000, 5235000, 4640000, 5606000),
    units: 'm',
    projection: "EPSG:31468",
};
```

In der Ebene werden wieder der Name, die URL des Servers und die Parameter in Form eines Objektes festgelegt. Im folgenden Beispiel wird eine Ebene für die Übersichtskarte im Maßstab 1 : 500 000 erstellt. Diese wird im WMS-Server mit *uk500* bezeichnet. Die Bilder werden im PNG-Format angefordert:

```
var uk = new OpenLayers.Layer.WMS(
    "UK 500",
    "http://www.geodaten.bayern.de/ogc/getogc.cgi?",
    {layers: 'uk500',
    format: 'image/png'}
);
```

Der im Folgenden verwendete WMS liefert je nach Maßstab das entsprechende Kartenwerk aus. Um eine optimale Auflösung im Client zu erhalten, ist es sinnvoll feste Zoomstufen einzuführen. Der hier benutzte WMS kann zwischen fünf Kartenwerken hin- und herschalten. Das sind zum einen die Topografischen Karten TK 25, TK 50, TK 100 und TK 200 und zum anderen die Übersichtskarte UK 500.

Die festen Zoomstufen kann man in den Kartenoptionen definieren. In einem Feld gibt man die von dem WMS vorgegebenen Bodenauflösungen [m/Pixel] in absteigender Reihenfolge an (vgl. OL Wiki 2008b):

```
Resolutions: [150,100,40,12.5,10,4,2.1166667],
```

Damit vermeidet man den Qualitätsverlust, der bei willkürlichen Zoomstufen bei rasterbasierten Datenquellen wie zum Beispiel WMS auftritt. In Abbildung 17 sieht man Darstellungen dieser Kartenwerke in *OpenLayers*.

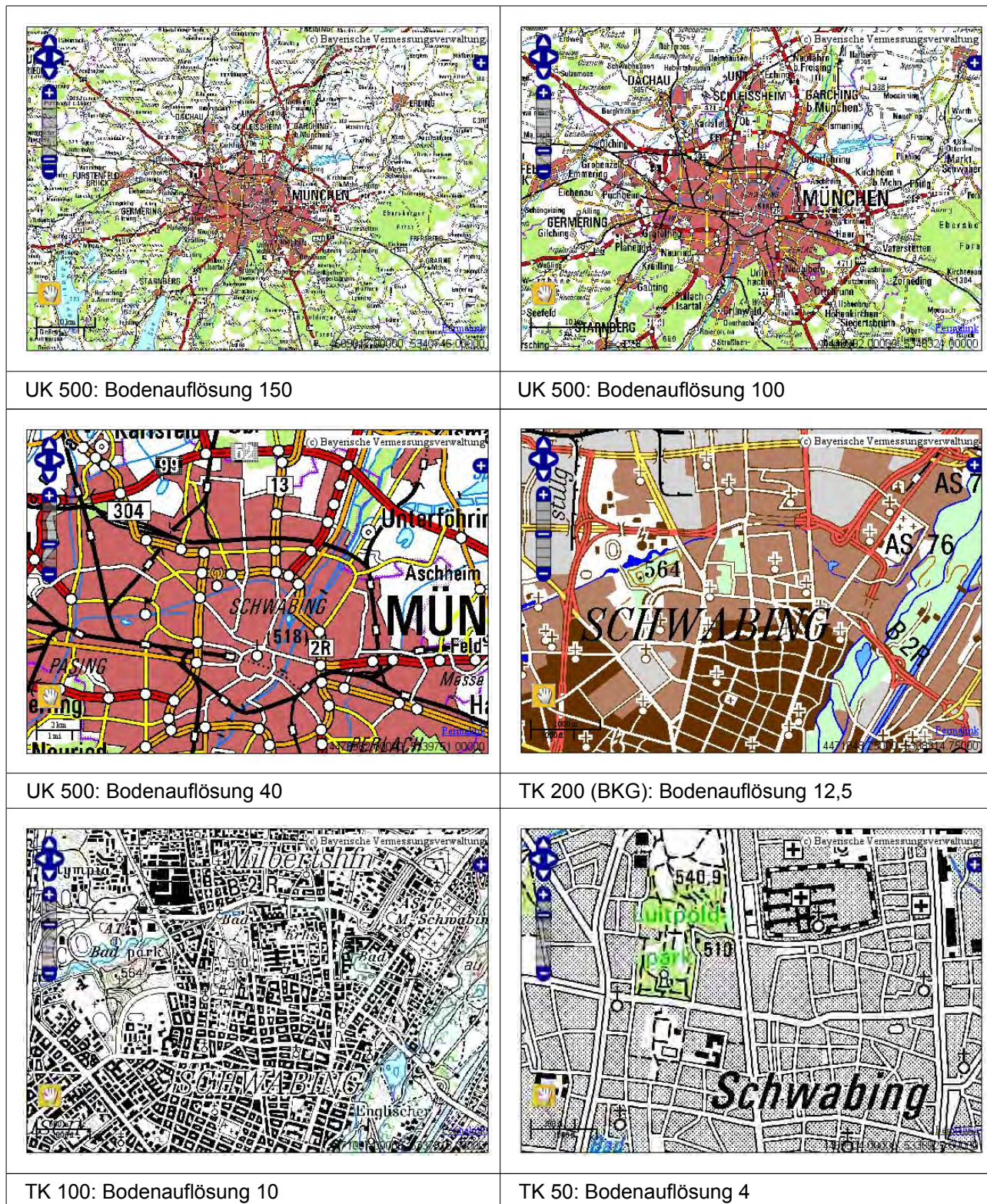


Abbildung 17: Kartenwerke des LVGs in OpenLayers

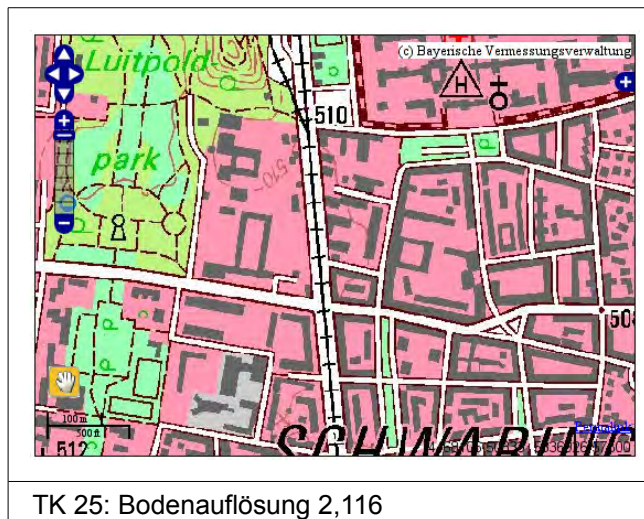


Abb. 17 (Forts.): Kartenwerke des LVGs in OpenLayers

Performance-Steigerung

Standardmäßig fordert *OpenLayers* die Daten ähnlich wie *Google Maps* kachelweise an (s. Abschnitt 4.1 I). Man kann aber auch einen herkömmlichen WMS-Viewer mit *OpenLayers* erstellen. Dazu setzt man beim Erstellen der *WMS*-Ebene das *singleTile*-Attribut auf *true* und *ratio* auf 1:

```
tk = new OpenLayers.Layer.WMS("TK",
    "http://geodaten.bvv.bybn.de/ogc/ogc_dipl.cgi?",
    {layers: 'tk', format: 'image/png'},
    {singleTile: true, ratio: 1});
```

Das Attribut *ratio* legt das Verhältnis der Größe der angeforderten Karte zum sichtbaren Bereich der Karte fest. Somit wird pro Kartenbild nur eine Datei in der Größe des Kartenbildes angefordert und dargestellt.

Die soeben im Konstruktor der *WMS*-Klasse gesetzten Parameter *singleTile* und *ratio* legen die vererbten Attribute der Klasse *Grid* fest. Die *WMS*-Klasse ist von der Klasse *Grid* abgeleitet (siehe Abbildung 18).

Verschiebt man den Kartenausschnitt, so wird erst nach dem Verschieben eine neue Karte geladen. Während des Vorgangs wird ein Teil der Karte nicht angezeigt und muss erst geladen werden. Dieses Verhalten veranschaulicht die Abbildung 19.

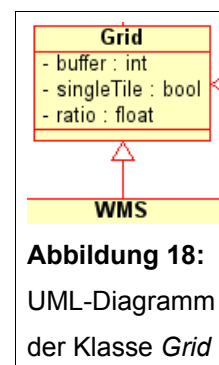


Abbildung 18:
UML-Diagramm
der Klasse *Grid*



Abbildung 19: WMS beim Verschieben der Karte

Setzt man den Wert *ratio* beispielsweise auf 2, so wird eine Karte geladen, die etwa doppelt so groß wie der sichtbare Bereich ist. Verschiebt man anschließend die Karte, so wird der im Voraus geladene Bereich sichtbar. Das macht den Vorgang dynamischer, vorausgesetzt die Anbindung an den WMS-Server ist gut. Dies ist eine Möglichkeit, das Verschieben für den Nutzer komfortabler zu gestalten. Wie diese Problematik von *Google Maps* und *OpenLayers* gelöst und damit eine Steigerung der Geschwindigkeit erreicht wurde, beschreibt das folgende Kapitel.

I. Kachelung von Google Maps

Google Maps ist die Browservariante von *Google Earth* und basiert auf der Ajax²³-Technologie. Nur die benötigten Daten, nicht wie früher die komplette Seite, werden im Hintergrund zwischen Browser und Server ausgetauscht. Diese asynchrone, sprich zeitversetzte, Übertragung der Daten macht eine erhöhte Interaktivität möglich.

Bei *Google Maps* ist das Kartenbild in mehrere Kacheln unterteilt. Diese werden einzeln aufgerufen. Nachdem der sichtbare Bereich geladen wurde, werden schon die umliegenden Kacheln vorgeladen. Falls der Benutzer dann die Karte verschieben will, sind die Daten im Randbereich bereits verfügbar. So kann das Kartenbild ohne Lücken dynamisch verschoben werden. Diese Eigenschaft wird *slippy Map*, zu Deutsch *flinke Karte*, genannt.

Da *Google Maps* nur eine proprietäre Schnittstelle besitzt, stellt sich die Frage, wie diese Funktionalität in einem offenen und standardkonformen System wie *OpenLayers* realisiert werden kann.

²³ Asynchronous JavaScript and XML ermöglichen das dynamische Nachladen von Internetseiten, s. Glossar

II. Kachelung in OpenLayers

Die Karte setzt sich aus vielen Kacheln zusammen (s. Abb. 20), die über mehrere WMS-Anfragen angefordert werden. Auf diese Weise können auch Kacheln außerhalb des sichtbaren Bereiches im Voraus geladen werden. So kann man einen dynamischen Kartenviewer erstellen, der *Google Maps* ähnelt.

Der schwarze Bereich in der Abbildung 20 entspricht dem angeforderten Kartenausschnitt. Um diesen mit Hilfe der Kacheln darstellen zu können,

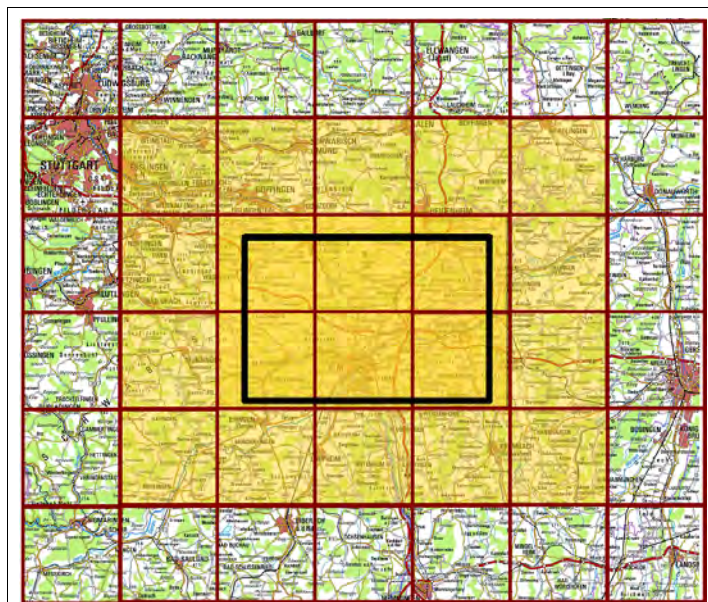


Abbildung 20: Kacheln eines Kartenausschnittes

müssen die sechs gelben Kacheln geladen werden. Die hellgelben Kacheln außerhalb des Anzeigebereiches werden schon im Voraus geladen, ohne dass eine Interaktion des Benutzers notwendig wäre. Dieser Vorgang basiert auf der *Ajax*-Technologie, mit der es möglich ist, Daten asynchron nachzuladen. Standardmäßig werden zwei Umringe geladen, wie in der Abbildung 20 gezeigt. Dadurch wird die Navigation flüssiger und dynamischer. Man nimmt in Kauf, dass das Laden der Kacheln eventuell unnötig war, falls der Benutzer die Karte nicht verschieben will.

In *OpenLayers* kann man in den Optionen des Kartenobjektes die Kachelfunktionalität einstellen. Dort wird die Größe der Kacheln festgelegt. Die Standardgröße sind 256 x 256 Pixel, in dem Beispiel wurde die Kachelgröße (*tileSize*) auf 500 x 500 Pixel festgelegt:

```
tileSize: new OpenLayers.Size(500,500)
```

Wie viele Kachelumringe zusätzlich, außerhalb des Anzeigebereiches, geladen werden, wird über die Variable *buffer* in den Ebenen eingestellt (vgl. OL Wiki 2008c):

```
new OpenLayers.Layer.WMS("TK", "http://geodaten.bvv.bybn.de/ogc/ogc_dipl.cgi?",
    {layers: 'tk', format: 'image/png'}, {buffer: 0});
```

Bisher wurden ausschließlich *Web Map Services* verwendet, deren Datenquelle im Kontext des LVG fertige Karten in Form von georeferenzierten Rasterdaten sind. Ein WMS kann aber auch Karten aus Vektordaten erstellen, indem er sich beispielsweise Polygone aus einer *PostGIS*²⁴-Datenbank holt, diese graphisch darstellt und beschriftet. Diesen WMS kann man wiederum in *OpenLayers* einbinden und anzeigen lassen.

Im Zusammenhang mit der in diesem Kapitel beschriebenen Kachelung muss man dann allerdings beachten, dass diese Methode nicht für alle Anwendungsbereiche sinnvoll ist. In der Abbildung 21 sieht man die doppelte und teilweise dreifache Beschriftung der Polygone an den Übergängen der Kacheln.

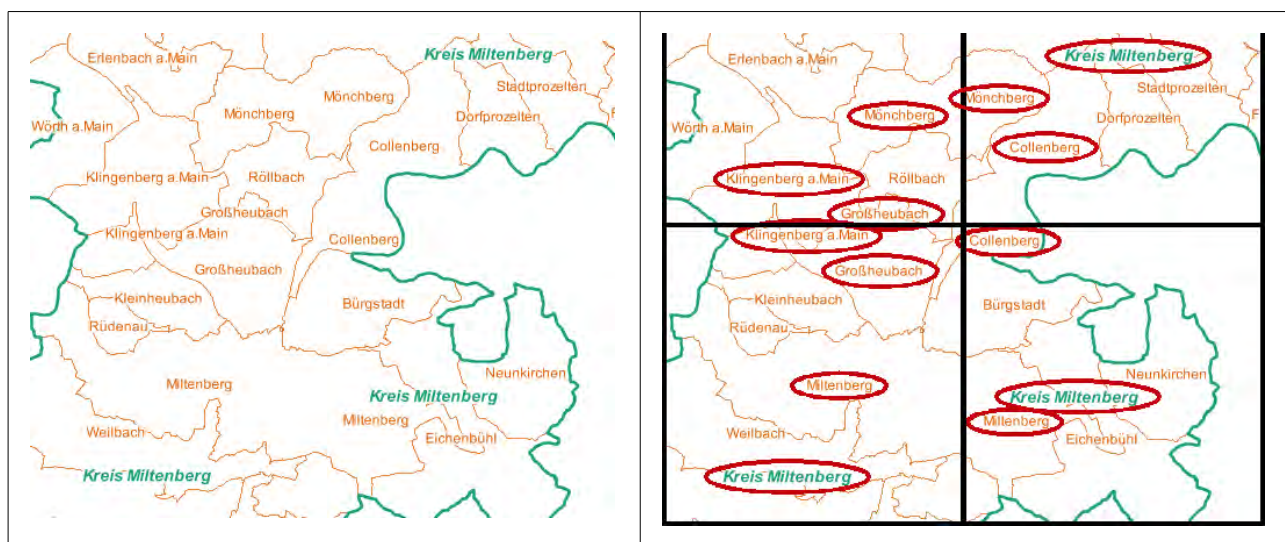


Abbildung 21: Kachelung eines WMS aus Vektordaten

Die Kacheln sind hier schwarz eingerahmt und die mehrfache Beschriftung ist mit Rot markiert. Innerhalb einer Kachel wird für jedes Polygon die Beschriftung im Zentrum platziert. Verläuft ein Polygon über mehrere Kacheln, so tritt der Bezeichnungstext mehrfach auf. Das liegt daran, dass ein WMS jede Kachel als unabhängige Anfrage behandelt und die Beschriftung möglichst optimal innerhalb der jeweiligen Geometrie positioniert. So werden Geometrieobjekte, die sich über mehrere Kacheln erstrecken auf jeder Kachel und damit mehrfach beschriftet.

In dieser Hinsicht ist die Kachelung eines WMS aus Vektordaten nicht sinnvoll. Eine Möglichkeit wäre, die Beschriftung von der Grafik zu separieren und in getrennten Ebe-

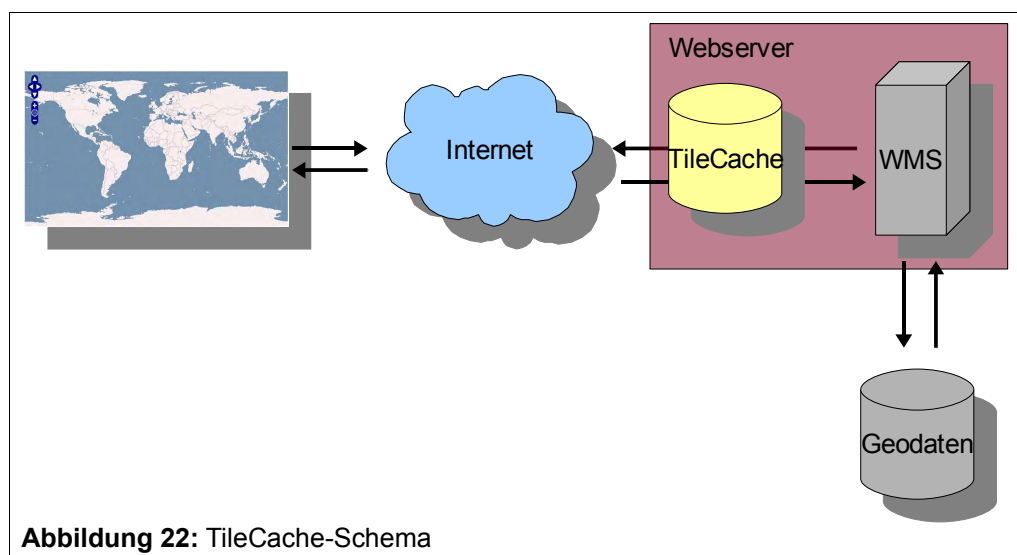
²⁴ Erweiterung der objektrelationalen Datenbank PostgreSQL um geografische Objekte, s. Glossar

nen, einmal gekachelt und einmal *en bloc*, anzuzeigen.

III. TileCache

Wenn die vielen einzelnen Kacheln vom WMS-Server erstellt werden müssen, führt das zu einer hohen Serverlast. Um den Rechenaufwand zu verringern, kann man alle Kacheln im Voraus generieren und auf einem Server speichern. Dazu nutzt man das Programm *TileCache*²⁵. Das Schema des *TileCaches* zeigt die Abbildung 22.

Auf der linken Seite sind der Webbrowser und Client dargestellt. Im Kartenviewer *OpenLayers*, der im Webbrowser ausgeführt wird, werden Daten aus dem Internet angezeigt, zum Beispiel die Karten eines WMS. Der WMS läuft auf einem Webserver und greift intern auf seine Geodaten zu. Der *TileCache* wird zwischen dem Internet und dem WMS ebenfalls auf einem Webserver platziert.



Die Anwendung *TileCache* wird von *MetaCarta* unter der BSD-Lizenz angeboten. *TileCache* kann von einem WMS-Server einen sogenannten Speicher (*Cache*) erstellen. Mit Hilfe des WMS bildet er gleichmäßige Kacheln, die er ablegt und für zukünftige Anfragen vorhält. *TileCache* ist also ein Kachelserver. Die Art und Weise der Kachelablage muss einheitlich sein. Die *OSGeo* hat bereits zwei Arten der Kachelung spezifiziert. Das ist zum einen die *WMS Tiling Client Recommendation* und zum anderen die *Tile Map Service (TMS) Specification* (vgl. *OSGeo* 2007). Der *WMS Tiling Client* wird auch als WMS-C be-

²⁵ www.tilecache.org [25.11.08]

zeichnet, das wiederum für *Web Map Service - Cached* steht. Diese beiden Spezifikationen werden von *TileCache* unterstützt. Clients, die ebenfalls TMS und WMS-C unterstützen, können auf den Kachelspeicher von *TileCache* zugreifen. Derzeit arbeitet das OGC an einem Standard mit dem Namen *Tiled WMS* (vgl. OGC 2007), der durch TMS angeregt wurde.

Nach der Installation von *TileCache* (vgl. *TileCache* 2008) muss die Konfigurationsdatei angepasst werden. Über sie wird auf das Programm *TileCache* zugegriffen. Die Datei heißt *tilecache.cfg* und befindet sich in dem Installationsordner. Unter *[cache]* kann man die Art und den Pfad des Caches einstellen. Der Parameter ist obligatorisch. Man kann die Kacheln auf der Festplatte des Servers speichern oder im Arbeitsspeicher. Es gibt aber auch die Möglichkeit, die Kacheln auf einem externen Webspeicher von *Amazon S3*²⁶ zu puffern. Der Arbeitsspeicher ist langsamer als der Festplattenspeicher, daher empfiehlt sich die Nutzung des Festplattencaches. Somit wird der Typ des Speichers als *Disk* festgelegt. Der Pfad zum Speicherordner wird unter *base* angegeben. Die Einstellungen sehen wie folgt aus:

```
[cache]
type=Disk
base=/home/wenge_ca/htdocs/tilecache
```

Als Nächstes wählt man die Ebene aus, die gecached werden soll und setzt deren Namen in eckige Klammern. Darunter folgen die Einstellungen. Handelt es sich um einen WMS, der gepuffert werden soll, so setzt man den Cachetyp auf *WMS*. Mit den weiteren Parametern legt man die URL, die Dateinamenserweiterung, die Bodenauflösungen, die Boundingbox, die Kachelgröße und die Projektion fest:

```
[tk]
type=WMS
url=http://geodaten.bvv.bybn.de/ogc/ogc_dipl.cgi?
extension=png
resolutions: 150,100,40,12.5,10,4,2.1166667
bbox = 4281000, 5235000, 4640000, 5606000
size=500,500
srs=EPSG:31468
```

Dabei ist zu beachten, dass die Einstellungen des *TileCaches* mit denen des *OpenLayers*-Kartenobjektes (*OpenLayers.Map*) übereinstimmen.

Die generelle Funktionsweise von *TileCache* besteht darin, WMS-Anfragen an ein vorde-

²⁶ Amazon Simple Storage Service dient der Datenspeicherung im Internet, s. Glossar

finiertes Gitter anzupassen. Damit werden immer die gleichen Kacheln angefordert, auf die dann zugegriffen wird. Da die Kacheln schon generiert sind, erreicht man damit eine Zeitersparnis.

TileCache lässt beim ersten Aufruf die Kacheln vom WMS-Server generieren und speichert sie ab. Ab dem zweiten Aufruf wird dann auf diese Kacheln zugegriffen. Man kann aber auch mit dem Programm *tilecache_seed.py* alle Kacheln auf einmal generieren.

tilecache_clean.py dagegen leert den Kachelspeicher. Dazu gibt man folgenden Befehl in die Konsole ein: `tilecache_clean.py -s0 /home/wenge_ca/htdocs/tilecache` Mit der Option `-s0` legt man die maximale *TileCache*-Größe auf 0 MB fest.

III.I. Visualisierung durch die WMS-Klasse

Das Einzige, was jetzt noch zu tun ist, ist eine Ebene der Klasse *WMS* anzulegen, deren URL auf das CGI²⁷-Skript des *TileCaches* zeigt. Diese Ebene muss dem Kartenobjekt hinzugefügt werden:

```
tc = new OpenLayers.Layer.WMS("TileCache",
    "http://localhost/tilecache-2.04/tilecache.cgi",
    {layers: 'tk', format: 'image/png', version: '1.1.1'});
```

III.II. Darstellung durch die TileCache-Klasse

OpenLayers stellt mit der Klasse *TileCache* eine Möglichkeit bereit, direkt auf diesen Datenspeicher zuzugreifen. Dazu muss man lediglich einen Namen, die URL des Datenspeichers, den Namen des Kartenwerkes und weitere Optionen als Parameter mitgeben:

```
tc2 = new OpenLayers.Layer.TileCache("Layer.TileCache",
    "http://localhost/tilecache",
    "tk",
    {format: 'image/png'});
```

Eine URL sieht dann beispielsweise so aus:

```
http://localhost/tilecache/tk/00/000/000/004/000/000/001.png
```

Genau mit dieser Ordnerstruktur liegen die Kacheln auch im *TileCache* vor. Diese Struktur begrenzt die Anzahl der Dateien pro Ordner auf maximal 1000 Stück. Das hat den Vorteil, dass die Dateisysteme ohne Weiteres mit Ordnern dieser Größe umgehen können.

²⁷ W3C-Standard für den Datenaustausch zwischen einem Webserver und dritter Software, s. Glossar

nen.

III.III. Anzeige durch die TMS-Klasse

Neben der *WMS*- und *TileCache*-Klasse existiert eine weitere Möglichkeit, die Kacheln eines *TileCaches* in *OpenLayers* einzubinden – die *TMS*-Ebene. Bei der Einbindung darf kein Fragezeichen am Ende der URL stehen.

```
tms = new OpenLayers.Layer.TMS(
    "TMS",
    "http://localhost/tilecache-2.04/tilecache.cgi/",
    {type: 'png', layername: 'tk'}
);
```

Nun werden die Kacheln vom Client wie folgt angefragt:

```
http://localhost/tilecache-2.04/tilecache.cgi/1.0.0/tk/0/4/1.png
```

TileCache überträgt die URL auf die tatsächlich verwendete Ordnerstruktur. Die *TMS*-Struktur wird also nur simuliert. Bei großen Zoomstufen würde die unbegrenzte Anzahl der Dateien dem Dateisystem Probleme bereiten. So wird die Ordnerstruktur von *TileCache* bevorzugt, da dort die Ordnerinhalte begrenzt sind (vgl. OL MailingList 2008c).

Soll der *TileCache* als WMS Benutzern zur Verfügung gestellt werden, die den Dienst in ihre Anwendung integrieren wollen, so können sie die benötigten Informationen mit Hilfe der *GetCapabilities*-Anfrage erhalten:

```
http://localhost/tilecache-2.04/tilecache.cgi?SERVICE=WMS&REQUEST=GetCapabilities
```

Der *GetCapabilities*-Request liefert die Fähigkeiten des WMS im XML-Format. Im Abschnitt *Capability* unter *VendorSpecificCapabilities* im Element *TileSet* findet man die notwendigen Angaben zu Projektion, Boundingbox, Bodenauflösungen, Kachelgröße, Format und Ebene. Siehe dazu Abbildung 23.

Mit dieser XML-Datei sind anscheinend keine Style-Informationen verknüpft. Nachfolgend wird die Baum-Ansicht des Dokuments angezeigt.

```

- <WMT_MS_Capabilities version="1.1.1">
+ <Service></Service>
- <Capability>
+ <Request></Request>
+ <Exception></Exception>
- <VendorSpecificCapabilities>
- <TileSet>
  <SRS>EPSG:31468</SRS>
  <BoundingBox SRS="EPSG:31468" minx="4281000.000000" miny="5235000.000000" maxx="4640000.000000"
  maxy="5606000.000000"/>
- <Resolutions>
  150.000000000000000000000000000000 100.000000000000000000000000000000 40.000000000000000000000000000000
  12.50000000000000000000000000000000 10.000000000000000000000000000000 4.000000000000000000000000000000 2.116666700000000012308
  </Resolutions>
  <Width>500</Width>
  <Height>500</Height>
  <Format>image/jpeg</Format>
  <Layers>tk</Layers>
  <Styles/>
</TileSet>
+ <TileSet></TileSet>
</VendorSpecificCapabilities>
<UserDefinedSymbolization SupportSLD="0" UserLayer="0" UserStyle="0" RemoteWFS="0"/>
+ <Layer></Layer>
</Capability>
</WMT_MS_Capabilities>

```

Abbildung 23: GetCapabilities-Antwort vom TileCache

III.IV. Performance-Test

Um den *TileCache* mit dem WMS hinsichtlich der Performance vergleichen zu können, wurde ein Test durchgeführt. Es wurden 100 Kacheln in einem Bereich von 10 x 10 Kacheln abgefragt. Dabei wurden jeweils die Antwortzeiten aufgezeichnet und der Mittelwert gebildet. In allen Ebenen von TK 25 bis UK 500 wurden die Antwortzeiten gemessen.

Die WMS-Anfragen werden im Schnitt innerhalb von 270 ms beantwortet. Die ersten Anforderungen an *TileCache* liegen minimal darüber, bei 290 ms. Das lässt sich durch das zusätzliche Speichern der Kacheln erklären. Die darauffolgenden Requests werden fast viermal so schnell beantwortet (70 ms). Somit spart man mit dem Einsatz von *TileCache* 75 Prozent der Zeit ein.

III.V. Gitterberechnung

Der Performance-Test wurde ohne Nutzung von *OpenLayers* durchgeführt. Die URLs zu den Kacheln wurden durch ein Programm generiert. Es wurde ein Rechts- und Hochwert als Startwert der Abfrage angegeben. Von dem Startwert aus wurden die Kacheln in Abständen der vorher festgelegten realen Kachelgröße abgerufen.

Beim Performance-Test trat der Fehler *Current x value 4463116.000000 is too far from tile corner x 4463033.336200* (Der gegenwärtige x-Wert 4463116 ist zu weit von der Kachelecke x 4463033,3362 entfernt) auf. Das bedeutet, dass die abzufragende Kachel zu weit von der im *TileCache* vorgehaltenen Kachel entfernt ist. Der Clou beim *TileCache* ist die Kartenanfragen auf ein bestimmtes Raster zu fixieren und dadurch immer auf die gleichen Kacheln zugreifen zu können. Aufgrund des aufgetretenen Fehlers wurde die Gitterberechnung des *TileCaches* untersucht. Die Art der Berechnung wurde durch Tests bestimmt. Das Gitter berechnet sich aus der maximalen Boundingbox, sprich der *maxExtent*-Variablen und der realen Kachelgröße (s. Gl. 2). Die beiden Werte *minx* und *miny* der *maxExtent*-Variablen legen den Ursprung fest. Die Gitterweite entspricht der Kachelgröße in der Natur. Die Abbildung 24 veranschaulicht die Gitterberechnung in *OpenLayers*.

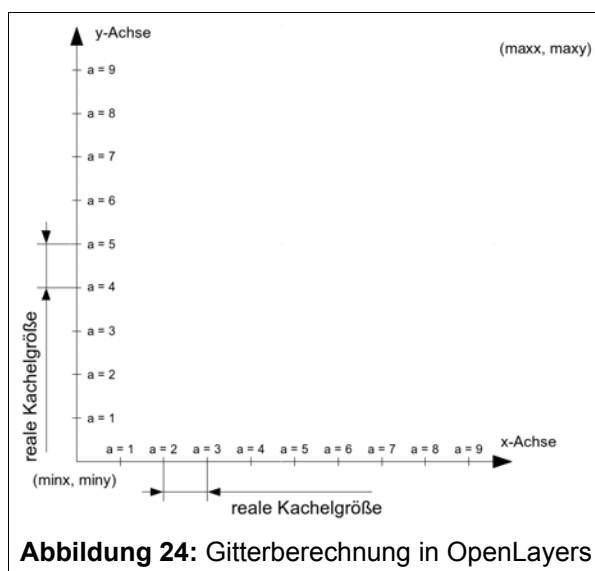


Abbildung 24: Gitterberechnung in OpenLayers

Die Berechnung der x- und y-Werte eines Gitterpunktes erfolgt nach den Gleichungen 4.1 und 4.2.

$$x = \text{minx} + (a \cdot \text{reale Kachelgröße} [m]) \quad (\text{Gl. 4.1})$$

$$y = \text{miny} + (a \cdot \text{reale Kachelgröße} [m]) \quad (\text{Gl. 4.2})$$

a Anzahl der Kacheln

Der Fehler konnte behoben werden, indem die Koordinate des Startwertes an das Gitter angepasst wurde.

IV. Direkter Zugriff

Wenn das darzustellende Kartenwerk im Originalzustand schon gekachelt vorliegt, kann man mit *OpenLayers* auch direkt auf die Daten zugreifen (Quelltext s. Anhang E). Dazu verwendet man die Klasse *TMS*. Da die gegebene Kachelstruktur in der Regel nicht der *TMS*-Spezifikation entspricht, muss man eine Funktion schreiben, die den Pfad zu den einzelnen Kacheln errechnet (vgl. OL Wiki 2008d). Mit ihr wird die URL dem Pfad zum Speicherort der Kacheln angepasst. Im vorliegenden Beispiel sind die Rasterdaten zuerst nach Ebenen, anschließend nach Bodenauflösungen und schließlich nach Rechtswerten in verschiedenen Ordnern abgelegt. Die Dateinamen setzen sich aus dem Rechts- und Hochwert zusammen. Beispielsweise sieht ein solcher Pfad folgendermaßen aus:

```
/tk50/400/4260000/42600005544000.png
```

Um diesen Pfad in *OpenLayers* generieren zu können, benötigt man einige Angaben. Die aktuelle Boundingbox wird der Funktion übergeben. Daraus benötigt man den linken und den unteren Wert, was mit dem Rechts- und Hochwert übereinstimmt. Die Bodenauflösung erhält man durch die Funktion *map.getResolution()*. Diese muss noch mit 100 multipliziert werden, da die Ordernamen der Bodenauflösung in Zentimetern entspricht. Aus diesen Werten kann man die benötigte URL zusammensetzen:

```
function get_url (bounds) {
    var res = this.map.getResolution();
    var x = bounds.left;
    var y = bounds.bottom;
    var z = res*100;

    var path = z + "/" + x + "/" + x + y + "." + this.type;
    var url = this.url;
    if (url instanceof Array) {
        url = this.selectUrl(path, url);
    }
    return url + path;
}
```

```
tms_uk500 = new OpenLayers.Layer.TMS(
    "TMS UK500",
    "http://localhost/uk500/",
    {type: 'png', getURL: get_url}
);
```

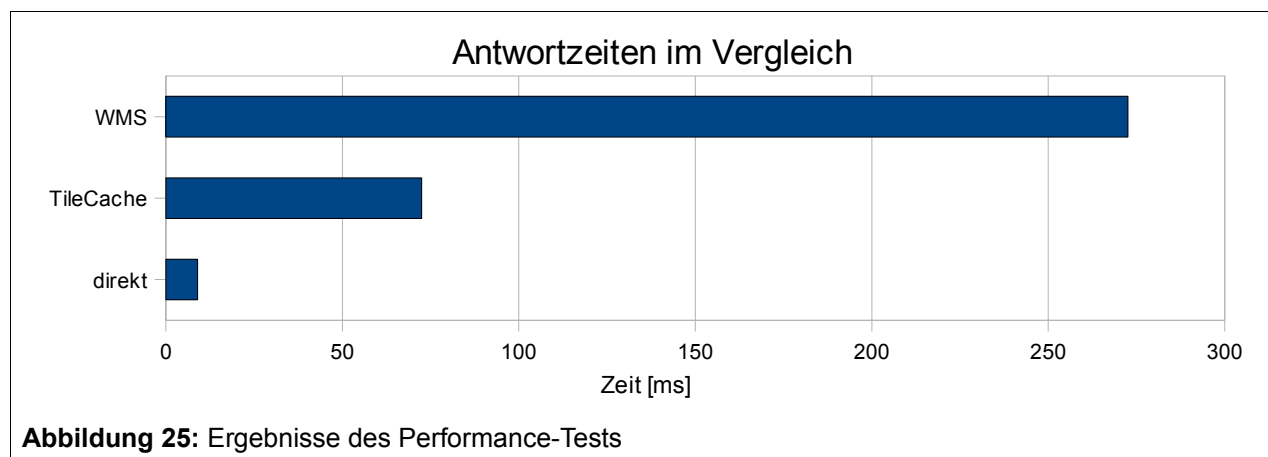
Um die gleiche Funktionalität wie beim vorher benutzten WMS zu gewährleisten, müssen die Ebenen je nach Zoomstufe ein- und ausgeblendet werden. Dafür wird das *zoomend*-Ereignis genutzt, um über die Bodenauflösung die *switch*-Anweisung zu steuern:

```

map.events.register("zoomend", map, function() {
switch(map.getResolution()) {
case 40:
map.setBaseLayer(tms_uk500);
break;
case 16:
map.setBaseLayer(tms_tk50_16);
break;
case 8:
map.setBaseLayer(tms_tk50_8);
break;
case 4:
map.setBaseLayer(tms_tk50_4);
break;
}});

```

In einem Test wurden wieder 100 Kacheln von den Ebenen UK 500 und TK 50 in den Bodenauflösungen 16, 8 und 4 m abgefragt. Der Performance-Test ergab, dass eine Kachel im Schnitt in weniger als 10 ms ausgeliefert wird. Somit wird im Vergleich zum *TileCache* nur ein Achtel der Zeit benötigt (s. Abb. 25). Im Vergleich zum WMS ist der direkte Datenzugriff 30-mal schneller. 96,7 Prozent der Zeit können eingespart werden.



Ein WMS muss selbst bei minimaler Verschiebung ein neues Kartenbild liefern. Das erhöht zusätzlich die ohnehin schon höhere Serverlast. Mit dem direkten Zugriff ist zwischen Benutzer und Daten kein WMS mehr geschaltet und die Dateien sind nicht wie beim *TileCache* in einer speziellen Größe gespeichert, sondern können direkt geladen werden. Die damit erreichte Geschwindigkeit ist beeindruckend. Aber es gibt auch einige Nachteile, die beachtet werden sollten. Zum einen unterliegt der direkte Zugriff keinem Standard. Die Zoomstufen sind ebenso wie beim *TileCache* vorgegeben. Des Weiteren können als Datenquelle ausschließlich rechteckige Bilddaten hinterlegt werden. Dies ist dann ungünstig, wenn die Bilddaten beispielsweise im Blattschnittformat vorliegen. Die

Kacheln müssen in einem zusätzlichen Arbeitsprozess erstellt werden. Dieser Schritt entfällt zum Beispiel beim Einsatz des UMN *MapServers*²⁸. Denn der *MapServer* kann unnötige Bereiche einer Bilddatei ausblenden. Trotz der genannten Nachteile ist das schnelle, lückenlose Verschieben im Internet zum Standard geworden.

V. WMS-Zugriff mit mehreren URLs

Nach Angabe von *OpenLayers* gibt es eine weitere Möglichkeit die Antwortzeiten zu optimieren (vgl. OL Wiki 2008c). Der Optimierungsvorschlag wird in diesem Abschnitt untersucht (Quelltext s. Anhang B).

Dazu muss vorweg der Begriff Domäne erklärt werden: Die Internetadresse des Host (*www.openlayers.org*) ist Bestandteil der URL (s. Abb. 16). Man unterteilt sie in Host (*www*), *Second Level Domain* (*openlayers*) und *Top Level Domain* (*org*). Die einzelnen Namensteile sind durch Punkte voneinander getrennt. Als *Subdomain* bezeichnet man eine Domäne (*domain*), die unter einer anderen liegt. So ist *openlayers* eine Subdomain von *org*.

Internetbrowser begrenzen die Anzahl von parallelen Aufrufen einer Domäne. Das rührt noch aus der Zeit als Wählverbindungen üblich waren. Damit die Verbindung nicht überlastet wird, lässt man maximal zwei parallele Anfragen pro Domäne zu. Erst wenn eine Anfrage abgeschlossen ist, wird die dritte Anfrage losgeschickt. Für Internetnutzer mit Breitbandverbindung ist das eine Einschränkung. Denn der Datenfluss wird unnötig gedrosselt und die Verbindung kann nicht in vollen Zügen genutzt werden.

In *OpenLayers* wird in der Regel das Kartenbild aus diversen Kacheln aufgebaut (s. 4.1 II). Es werden also viele Bilder von ein und derselben Domäne angefordert. Um eine Breitbandverbindung vollständig nutzen zu können, fordert man über verschiedene URLs die Daten an. So können beispielsweise bei fünf URLs zehn Bilder gleichzeitig heruntergeladen werden. Wenn man beim Anlegen einer neuen Ebene in *OpenLayers* statt einer URL mehrere angibt, so werden die Kacheln von allen URLs parallel angefordert:

28 Open-Source-Software zur Darstellung raumbezogener Daten, s. Glossar

```
tk_url = new OpenLayers.Layer.WMS("TK URLs",
    ["http://geodaten.bvv.bybn.de/ogc/ogc_dipl.cgi?",
     "http://url1/ogc/ogc_dipl.cgi?",
     "http://url2/ogc/ogc_dipl.cgi?",
     "http://url3/ogc/ogc_dipl.cgi?",
     "http://url4/ogc/ogc_dipl.cgi?"],
    {layers: 'tk', format: 'image/png'});
```

In diesem Beispiel wurden die verschiedenen URLs in die *hosts*-Datei eingetragen. Die Datei *hosts* ist eine lokale Textdatei, die im Betriebssystem der festen Zuordnung von Hostnamen zu IP-Adressen dient. Normalerweise löst man das über unterschiedliche *Subdomains*, zum Beispiel *url1.geodaten.bvv.bybn.de* und *url2.geodaten.bvv.bybn.de*.

In einem Performance-Test wurde jeweils auf dasselbe Kartenwerk zugegriffen, einmal mit einer URL und das andere mal mit fünf. Dabei wurden die Antwortzeiten gemessen und die Anzahl der angefragten Kacheln notiert. Das Ergebnis zeigt zwar, dass ein Kartenbild mit Hilfe von fünf URLs minimal schneller (wenige Zehntelsekunden) aufgebaut wird, aber nach Untersuchung der angefragten Kacheln, ist diese minimale Steigerung der Geschwindigkeit auf die niedrigere Zahl der Kacheln zurückzuführen. Im Schnitt wird sogar eine Kachel über eine URL minimal schneller (wenige Hundertstelsekunden) ausgeliefert. Leider konnte die erhoffte Steigerung der Geschwindigkeit in dem eingesetzten Testszenario nicht nachgewiesen werden. Dennoch sollte die Idee von Webdiensteanbietern in Betracht gezogen und im Einzelfall überprüft werden.

4.2 WFS

Neben den *Web Map Services* zählen auch die *Web Feature Services* (WFS) zu den Online-Diensten, die vom LVG angeboten werden. Der WFS recherchiert nach Geodaten und stellt die Ergebnisse in Form einer GML-Datei bereit. Die *Geography Markup Language* (GML)²⁹ dient der Modellierung, dem Transport und der Speicherung von raumbezogenen Objekten.

Der WFS beschränkt sich ausschließlich auf Vektordaten, wie sie in Datenbanken abgelegt werden können. Objekte (*Features*) können nach räumlichen oder inhaltlichen Bedingungen ermittelt werden. Wie beim WMS gibt es auch beim WFS die *GetCapabilities*-Anfrage, die über die Fähigkeiten des WFS Auskunft gibt und die Objektarten (*Feature Types*) beinhaltet. Mit *DescribeFeatureType* kann die Objektstruktur einzelner Objekte

²⁹ s. Glossar

ten (*Feature Types*) abgefragt werden. Mit *GetFeature* kann man schließlich die Abfrage durchführen und erhält Objekte (*Features*) als Antwort.

Der folgende Abschnitt schildert die Einbindung eines WFS in *OpenLayers*. In den bisherigen Beispielen der Implementierung von externen Rasterdatenquellen zum Beispiel über WMS wurde auf das *Image-Tag* von HTML zurückgegriffen, mit dessen *Source*-Angabe man auf das entsprechende Bildmaterial zugegriffen hat. Dabei gibt es keinerlei Einschränkungen. Anders ist das bei GML-Vektordaten, die über *JavaScript* eingebunden werden. Bei *JavaScript* ist aus Sicherheitsgründen der Zugriff auf andere Domänen unterbunden. Wenn man mit Hilfe von *OpenLayers* auf einen *Web Feature Service* zugreifen will, der auf einer anderen Domäne liegt (das entspricht dem Regelfall), erhält man eine Fehlermeldung. Um den Zugriff dennoch möglich zu machen, verwendet man einen kleinen Trick. Man greift nicht direkt auf eine andere Domäne zu, sondern auf ein CGI-Skript, das in der eigenen Domäne liegt. Das CGI-Skript leitet die Anfragen dann auf die andere Domäne weiter und schickt die erhaltene Antwort zurück. Auf der Homepage von *OpenLayers* steht ein solches CGI-Skript zum Herunterladen bereit. Nachdem in diesem Proxy-Skript (engl. Stellvertreter, s. Anhang F) unter *allowedHosts* der WFS-Server eingetragen ist und in *OpenLayers* dem *ProxyHost* die URL zum Skript zugewiesen wurde, ist dieses Problem umgangen.

Im proxy.cgi-Skript: `allowedHosts = ['geoentw.bvv.bayern.de:8080']`

In *OpenLayers*: `OpenLayers.ProxyHost = "/cgi-bin/proxy.cgi?url=";`

4.2.1 Darstellung von Punktobjekten

Ein WFS kann Punkte oder Polygone liefern. Um die Punkte anzuzeigen, benötigt man einen Marker. Als Marker kann man zum Beispiel ein kleines Bild oder einen Punkt verwenden. Zunächst wird ein Marker mit einem Icon definiert:

```
var symbolizer_blue = OpenLayers.Util.applyDefaults(
    {externalGraphic: "./img/marker-blue.png",
      pointRadius: 12,
      fillOpacity: 1},
    OpenLayers.Feature.Vector.style["default"]
);
```

Einen Punktmarker kann wie folgt erstellt werden:

```
var point_symbolizer = OpenLayers.Util.applyDefaults({pointRadius: 6,
  fillColor: '000000', fillOpacity: 1,
  strokeWidth: 7, strokeColor: '#329cff'},
  OpenLayers.Feature.Vector.style["default"]);
```

Mit den Parametern *pointRadius*, *strokeWidth*, *strokeColor*, *fillColor* und *fillOpacity* kann man den Radius, die Strichstärke und die Farbe des Kreises sowie die Farbe und Deckkraft der Füllung festlegen.

Nun können die Ebenen für die Objekte des WFS angelegt werden. Beim Anlegen der WFS-Ebene werden Name, URL, *Typename* und *styleMap* bestimmt. Über *styleMap* findet die Zuordnung zu den erstellten Markern statt. *Typename* bestimmt die Objekte, die in der Ebene dargestellt werden sollen:

```
var wfs2 = new OpenLayers.Layer.WFS("WFS Regierungsbezirke",
  "http://geontw.bvv.bayern.de:8080/geoserver/wfs",
  {typename: 'rbezirke_wfs'},
  {styleMap: new OpenLayers.StyleMap({"default": symbolizer_gold})});
```

In diesem Beispiel wurden mit Hilfe eines WFS Regierungsbezirke, Landkreise Gemeinden und Gemeindeteile visualisiert. Das Ergebnis sieht man in der Abbildung 26.

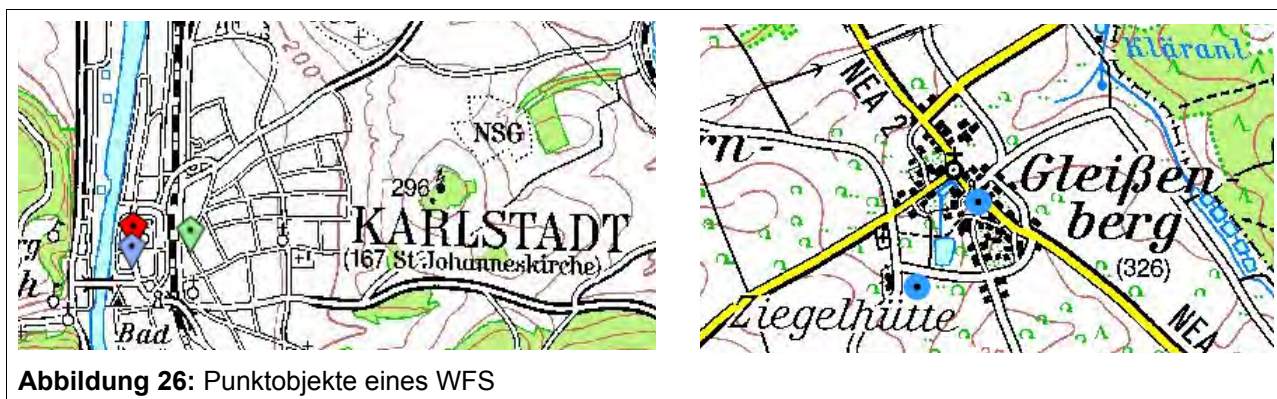


Abbildung 26: Punktobjekte eines WFS

Da die beiden Ebenen *Gemeinde* und *Gemeindeteile* die meisten Punktobjekte liefern und der Browser mit der Darstellung so vieler Marker Probleme hat, werden die genannten Ebenen erst ab einer Auflösung von 10 m/Pixel eingeblendet. Dazu setzt man die Attribute *minResolution* und *maxResolution* in den Optionen der Ebene:

```
var wfs4 = new OpenLayers.Layer.WFS("Name", "URL", Parameter,
  {minResolution: 2.1166667, maxResolution: 10});
```

Die Ebenen werden auf diese Weise immer in der Legende angezeigt. Sie werden allerdings bei höheren Bodenauflösungen in der Legende grau dargestellt.

Weitere Informationen zu den einzelnen Punktobjekten erhält man, wenn das Attribut *extractAttributes* auf *true* gesetzt ist. Anschließend kann man auf die Objekteigenschaften zugreifen und sie über Popups ausgeben. Diese Eigenschaft sollte aber nur dann eingestellt werden, wenn die Daten auch benötigt werden. Denn so dauert das Bearbeiten der Anfrage länger.

```
var wfs1 = new OpenLayers.Layer.WFS("Name", "URL", Parameter,
    {extractAttributes: true, style: style1});
```

Das Steuerelement *SelectFeature* macht man sich zu Nutze, um die Objekteigenschaften anzuzeigen. Als ersten Parameter wird die Ebene angegeben für die das Steuerelement aktiviert werden soll. In den Optionen wird unter anderem das Popup generiert. *Hover: true* bedeutet, dass ein Objekt dann selektiert ist, wenn der Benutzer mit der Maus darüberfährt. Beim Selektieren (*onSelect*) wird das Popup erstellt. Die Parameter des Popups setzen die ID, die Koordinaten, die Größe, den Inhalt, den Anker und die Schaltfläche zum Schließen fest. Die Koordinaten erhält man von der Objektgeometrie. Der Inhalt des Popups kann im HTML-Format angegeben werden. Die Objekteigenschaften kann man direkt mit dem Namen ansprechen. Schließlich muss das Popup noch der Karte hinzugefügt werden. Beim Deselektieren (*onUnselect*) wird das Popup geschlossen.

```
var select = new OpenLayers.Control.SelectFeature(Ebene, options);
var options = {hover: true,
  onSelect: function(feature){popup = new
    OpenLayers.Popup.FramedCloud("WFS",
      feature.geometry.getBounds().getCenterLonLat(),
      null,
      "Landkreis: <b>" + feature.attributes.lkr + "</b><br>
      Regierungsbezirk: <b>" + feature.attributes.rbez + "</b>",
      null,
      false),
    map.addPopup(popup)},
  selectStyle: style1_selected,
  onUnselect: function(feature){selectStyle: style1,
    popup.destroy()}
}
```

Mit *selectStyle* wird jeweils die Darstellung im selektierten bzw. deselektierten Zustand definiert. Damit wird für den Benutzer deutlich, dass hinter den Objekten weitere Informationen abrufbar sind.

```
var style1 =
  OpenLayers.Util.extend({}, OpenLayers.Feature.Vector.style['default']);
style1.strokeWidth = 2;
style1.strokeStyle = "#ff0000";
style1.fillOpacity = 1;
```



```
var style1_selected =
  OpenLayers.Util.extend({}, OpenLayers.Feature.Vector.style['default']);
style1_selected.strokeWidth = 2;
style1_selected.strokeColor = "#ffff00";
style1_selected.fillOpacity = 1;
```

Das Steuerelement zum Selektieren muss schließlich noch der Karte hinzugefügt und aktiviert werden:

```
map.addControl(select);
select.activate();
```

In der Abbildung 27 sieht man beispielhaft ein Punktobjekt des Landkreises Miltenberg. In dem Popup werden die Informationen zu Landkreis und Regierungsbezirk angezeigt. Den Quelltext im Zusammenhang findet man im Anhang G.



Abb. 27: Popup eines WFS-Punktobjektes

4.2.2 Darstellung von Polygonen

Abgesehen von der Möglichkeit der Darstellung von WFS-Punktobjekten in *OpenLayers* können zudem WFS-Polygone visualisiert werden. Polygone erhält man, indem man bei der Erstellung einer WFS-Ebene unter *typename* den Namen einer Polygonsammlung angibt. Hier muss man wieder die Darstellungsfähigkeit des Browsers beachten. Um ihn nicht zu überlasten und die Übersichtlichkeit zu bewahren, werden die Polygone erst bei niedriger Auflösung eingeblendet:

```
var wfs_poly = new OpenLayers.Layer.WFS("WFS Polygone",
  "http://geontw.bvv.bayern.de:8080/geoserver/wfs",
  {typename: 'v_shape_axflst'},
  {maxResolution: 0.25, minResolution: 0.1});
```

In dem Beispiel sollen die Flurstücke angezeigt werden, als Hintergrund dient ein WMS auf die *Digitale Flurkarte*. Wie beim Typ Punkt muss wieder ein Steuerelement *SelectFeature* angelegt werden, um die Polygone auswählbar zu machen:

```
var select_poly = new OpenLayers.Control.SelectFeature(wfs_poly);
map.addControl(select_poly);
```

Da die Ebene der Flurstücke überlappungsfrei und lückenlos ist, kann man überall mit

der Maus Polygone auswählen, aber man kann mit der Maus die Karte nicht mehr verschieben. Um diese Funktion wiederherzustellen wird ein Schalter benötigt, mit dessen Hilfe man zwischen Navigation und Selektion wechseln kann. Da im *Framework OpenLayers* eine solche Klasse nicht vorhanden ist, wird eine Neue erstellt.

Die Klasse *Toggle* ist von der Klasse *Control* abgeleitet. Als Typ des Steuerelementes wird *Toggle* ausgewählt. Der Schalter kann somit durch einen Klick aktiviert bzw. deaktiviert werden. Beim Aktivieren der Schaltfläche wird das *SelectFeature*-Steuerelement aktiviert, beim Deaktivieren wird auch das Steuerelement deaktiviert. Der Schalter schaltet also die Selektionsfunktion ein und aus.

Im Folgenden ist der Quelltext zur Klasse *Toggle* abgebildet:

```
/**
 * programmed by Carolin Wengerter
 *
 * @requires OpenLayers/Control.js
 *
 * Class: OpenLayers.Control.Toggle
 * A toggle control for use with <OpenLayers.Control.Panel>.
 *
 * Inherits from:
 * - <OpenLayers.Control>
 */
OpenLayers.Control.Toggle = OpenLayers.Class(OpenLayers.Control, {
  /**
   * Property: type
   * {Integer} OpenLayers.Control.TYPE_TOGGLE.
   */
  type: OpenLayers.Control.TYPE_TOGGLE,
  /**
   * Method: activate
   * Explicitly activates a control and it's associated
   * handler if one has been set. Controls can be
   * deactivated by calling the deactivate() method.
   *
   * Returns:
   * {Boolean} True if the control was successfully activated or
   *           false if the control was already active.
   */
  activate: function () {
    if (this.active) {
      return false;
    }
    if (this.handler) {
      this.handler.activate();
    }
    this.active = true;
    this.events.triggerEvent("activate");
    select_poly.activate();
    return true; },
```

```

/**
 * Method: deactivate
 * Deactivates a control and it's associated handler if any. The exact
 * effect of this depends on the control itself.
 *
 * Returns:
 * {Boolean} True if the control was effectively deactivated or false
 *           if the control was already inactive.
 */
deactivate: function () {
    if (this.active) {
        if (this.handler) {
            this.handler.deactivate();
        }
        this.active = false;
        this.events.triggerEvent("deactivate");
        select_poly.deactivate();
        return true;
    }
    return false;
},
CLASS_NAME: "OpenLayers.Control.Toggle"
});

```

Anschließend wird die Datei mit dem Namen *Toggle.js* unter *OpenLayers/Control* gespeichert und in die Datei *OpenLayers.js* eingetragen. In der Datei *OpenLayers.js* gibt es die Variable *jsfiles*, die einen Array sämtlicher *JavaScript*-Dateien der *OpenLayers*-Bibliothek enthält. Diesem Array fügt man den Eintrag *"OpenLayers/Control/Toggle.js"*, hinzu. Nun muss der Schalter im Programmcode noch erzeugt werden. Dabei wird durch *title* die Kurzinfo und durch *displayClass* der CSS³⁰-Style festgelegt:

```
var select_toggle = new OpenLayers.Control.Toggle({
    displayClass: "mbControlSeect", title: "Selektion"});
```

Abschließend muss die Funktionsleiste erstellt werden, in die der Schalter aufgenommen wird. Sie wird dann der Karte als Steuerelement hinzugefügt:

```
var panel = new OpenLayers.Control.Panel({displayClass: "olControlPanel"});
panel.addControls([select_toggle]);
map.addControl(panel);
```

Die *style.css*-Datei steuert das Layout. Die Position der Funktionsleiste wird im Abschnitt *.olControlPanel* bestimmt und die Größe in *.olControlPanel div*:

```
.olControlPanel {
    top: 200px;
    left: 14px;
}
```

30 CSS dient der Formatierung von HTML-Seiten, s. Glossar

```
.olControlPanel div {
  width: 24px;
  height: 22px;
}
```

Außerdem wird das Symbol des aktivierten und des deaktivierten Schalters festgelegt:

```
.olControlPanel .mbControlSelectItemActive {
  background-image: url("img/draw_polygon_on.png");
  background-repeat: no-repeat;
}

.olControlPanel .mbControlSelectItemInactive {
  background-image: url("img/draw_polygon_off.png");
  background-repeat: no-repeat;
}
```

Die Abbildung 28 zeigt den erstellten Schalter und ein selektiertes Polygon.

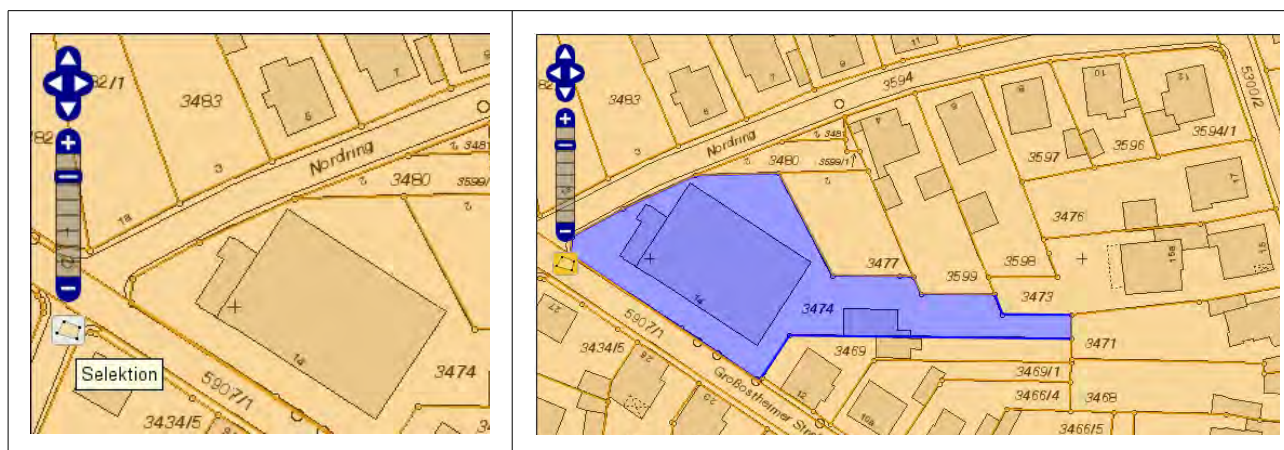


Abbildung 28: Schalter und Funktion zur Selektion von Polygonen

Der Quelltext zu den visualisierten WFS-Polygonen befindet sich im Anhang H.

4.3 GeoRSS

4.3.1 Allgemein

OpenLayers bietet vielerlei Möglichkeiten Geodaten zu visualisieren, sei es nun von *Google*, *Yahoo* (s. Kapitel 5) oder von einem *Web Map Service* oder Objekte eines *Web Feature Services* (s. Abschnitt 4.1 und 4.2). Außerdem kann man mit Hilfe von *OpenLayers* Daten im Format *GeoRSS*³¹ anzeigen. Wie diese Funktionalität in *OpenLayers* verfügbar gemacht werden kann, beschreibt dieses Kapitel (Quelltext s. Anhang I).

³¹ georss.org [26.11.08]

Wie beim WFS kommt auch bei *GeoRSS* GML zur Anwendung. GML wird durch ein XML-Schema, Dokument oder auch durch beides logisch eingeschränkt. Das nennt man *GML-Profil*. RSS ist ein solches *GML-Profil*. Der RSS ist ein Webdienst, ähnlich einem Newsticker. Man kann einen sogenannten *RSS-Feed* abonnieren und wird dann automatisch über neue Einträge informiert. *GeoRSS* ist ein RSS für geografisch kodierte Objekte. Es gibt zwei Kodierungen von *GeoRSS*: *Simple* und GML. Mit *Simple* können Punkte, Linien, Rechtecke und Polygone nur im WGS84-Koordinatensystem dargestellt werden. GML unterstützt dagegen auch andere Bezugssysteme.

Beispiel zur Punktdarstellung in *GeoRSS simple*:

```
<georss:point>45.256 -71.92</georss:point>
```

und in *GeoRSS GML*:

```
<georss:where>
  <gml:Point>
    <gml:pos>45.256 -71.92</gml:pos>
  </gml:Point>
</georss:where>
```

Einen *RSS-Feed*, der *GeoRSS*-Objekte liefert, nennt man *Geo-Feed*. Geografisch kodierte Objekte eines *Geo-Feeds* können zum Beispiel Verkehrsnachrichten, aktuelle Erdbeben oder georeferenzierte Fotos der Internetdatenbank *Flickr*³² sein.

4.3.2 Visualisierung von georeferenzierten Objekten

Die georeferenzierten Objekte eines *Geo-Feeds* können auf einer Karte angezeigt werden. Für die Visualisierung kann man *OpenLayers* nutzen. In diesem Abschnitt werden zwei verschiedene *Geo-Feeds* visualisiert. Zunächst wird der *Geo-Feed* eines Staumelders durch die Klasse *GeoRSS* eingebunden und anschließend wird die GML-Klasse zur Darstellung von georeferenzierten Fotos genutzt.

Da beim Zugriff auf die *Geo-Feeds* die Daten von einem fremden Server geladen werden müssen, wird wieder auf den *Proxy* (s. Abschnitt 4.2) zurückgegriffen. Dazu muss im Quelltext ein Proxyhost angegeben werden:

```
OpenLayers.ProxyHost = "/cgi-bin/proxy.cgi?url=";
```

In dieses Skript müssen die anzufragenden Hosts (*www.antenne.de* und *api.flickr.com*)

³² www.flickr.com [25.11.08]

eingetragen werden.

4.3.2.1 Visualisierung durch die Klasse GeoRSS

Der Radiosender *Antenne Bayern* bietet ein Abonnement der Staumeldungen für Deutschland in Form eines *Geo-Feeds* an. Dieser soll in *OpenLayers* über die Klasse *GeoRSS* dargestellt werden. Die Ebene wird über den Konstruktor der Klasse *GeoRSS* erstellt. Der Aufbau des Konstruktors gestaltet sich wie üblich. Zunächst wird der Name der Ebene angegeben, der in der Legende angezeigt werden soll und anschließend folgt die URL:

```
var georssstau = new OpenLayers.Layer.GeoRSS("Name",
    "http://www.antenne.de/cf/verkehr/deutschland_rss.xml");
```

Damit wird für jedes georeferenzierte Objekt ein Standard-Marker und -Popup generiert.

Die Abbildung 29 zeigt die Visualisierung des *Geo-Feeds* in *OpenLayers*. Die Staumeldungen werden zunächst durch Marker dargestellt, ein Mausklick öffnet das entsprechende Popup mit Zusatzinformationen und einem Link.



Abb. 29: GeoRSS-Ebene mit Staumeldungen

4.3.2.2 Visualisierung durch die Klasse GML

Eine weitere Möglichkeit, GeoRSS-Objekte in *OpenLayers* zu visualisieren, ist über die Klasse *GML*. Diese bietet eine größere Flexibilität zur Gestaltung der Marker und Popups als die Klasse *GeoRSS*. In diesem Abschnitt wird ein *Geo-Feed* von *Flickr* in *OpenLayers* eingebunden.

Flickr ist eine Fotoplattform im Internet, die von *Yahoo* betrieben wird. Benutzer können sich kostenlos anmelden und ihre digitalen Bilder hochladen. Den Bildern können sie Kommentare und Notizen mitgeben. Die Fotos kann man anderen Benutzern zur Verfügung stellen. So bleiben die Benutzer untereinander in Kontakt und werden über neue

Bilder ihrer Freunde und Bekannte informiert. Die Fotos können mit sogenannten *Geotags*, die die Koordinaten des Aufnahmeortes beinhalten, versehen werden. Der *Geotag* beinhaltet die Koordinaten des Aufnahmeortes.

Die URL zu einem *Geo-Feed* von *Flickr* sieht wie folgt aus:

```
http://api.flickr.com/services/feeds/geo/?lang=de-de&format=rss2
```

Über die mit einem Fragezeichen (?) angehängten und mit einem Und-Zeichen (&) verbundenen Abfrageparameter kann man Format (`format=rss2`) und Sprache (`lang=de-de`) festlegen. Das heißt, es wird eine beliebige Auswahl von georeferenzierten Bildern abgefragt. Die Kollektion der Bilder ändert sich von Aufruf zu Aufruf.

In dem Beispiel werden zunächst die Fotos in Miniaturgröße angezeigt (vgl. OL Bsp 2008c). Beim Anklicken werden sie vergrößert in einem Pop-up-Fenster mit Titel und Beschreibung visualisiert. Das Pop-up wird geschlossen, sobald man ein anderes Bild auswählt oder auf die Karte klickt.

Eine neue Ebene der Klasse *GML* wird angelegt: Neben Name und URL wird noch das Format GeoRSS angegeben. In den Formatoptionen (*formatOptions*) wird das Miniaturbild zur Anzeige der geografisch kodierten Objekte auf der Karte festgelegt. Mit *styleMap* legt man die Vergrößerung des Miniaturbildes bei Selektion fest:

```
var markerLayer = new OpenLayers.Layer.GML("Fotos von Flickr",
    "http://api.flickr.com/services/feeds/geo/?lang=de-de&format=rss2",
    {format: OpenLayers.Format.GeoRSS,
     formatOptions: {
       createFeatureFromItem: function(item) {
         var feature =
OpenLayers.Format.GeoRSS.prototype.createFeatureFromItem.apply(this, arguments);
         feature.attributes.thumbnail =
this.getElementsByTagNameNS(item, "*", "thumbnail")[0].getAttribute("url");
         return feature;}},
     styleMap: new OpenLayers.StyleMap({
       "default":
         new OpenLayers.Style({externalGraphic: "${thumbnail}", pointRadius: 25}),
       "select": new OpenLayers.Style({pointRadius: 40})
     })
    });
map.addLayer(markerLayer);
```

Nachdem die Ebene mit obigem Quelltext erstellt wurde, zeigt sie die Miniaturbilder, die durch Selektion vergrößert werden. Um noch weitere Informationen zu dem Bild zu erhalten, wird mit den folgenden Codezeilen ein Pop-up erzeugt. Das Pop-up wird durch einen Mausklick auf das Miniaturbild geöffnet. Um dieses Klickereignis abzufangen, benö-

tigt man das *SelectFeature*-Steuerelement:

```
var popup;
var popupControl = new OpenLayers.Control.SelectFeature(markerLayer, {
    onSelect: function(feature) {var pos = feature.geometry;
        if (popup) {
            map.removePopup(popup);
        }
        popup = new OpenLayers.Popup.FramedCloud(
            "popup",
            new OpenLayers.LonLat(pos.x, pos.y),
            null,
            "<b>" + feature.attributes.title + "</b>" + feature.attributes.description,
            null,
            false);
        map.addPopup(popup);
    },
    onUnselect: function(feature) {popup.destroy();}});
map.addControl(popupControl);
popupControl.activate();
```

Anschließend wird das Popup generiert und gestaltet. Im Popup sollen Titel und Beschreibung des Fotos abgebildet werden. Das Ergebnis zeigt Abbildung 30.



Abbildung 30: GML-Ebene mit Fotos von Flickr

4.3.3 Mögliche Anwendungsgebiete

Mit GeoRSS könnte man zum Beispiel die Homepage einer Schlösserverwaltung gestalten. Auf einer Karte des Bundeslandes kann man die Fotos der historischen Gebäude abbilden. Durch einen Klick wird man dann beispielsweise auf die Informationsseite eines bestimmten Schlosses weitergeleitet.

Auch wäre es mit der in diesem Abschnitt beschriebenen Technik denkbar, Informationen zur Leitungsdokumentation zu präsentieren. Da dort Fotos von Leitungskästen und Hochspannungsmasten benötigt werden, könnten diese über eine Karte zugänglich gemacht werden.

Ein weiteres Anwendungsgebiet könnte auch der Denkmalschutz sein. Das Amt für Denkmalpflege dokumentiert die Denkmäler unter anderem mit Bildern. Eine Möglichkeit der Visualisierung mit Hilfe von *OpenLayers* ist vorstellbar.

Zudem besteht großes Interesse an der Visualisierung von georeferenzierten Objekten seitens der Immobilienwirtschaft.

5 Integration von proprietären Geodaten

Das fünfte Kapitel befasst sich mit der Einbindung von proprietären Geodaten. Dafür werden von *OpenLayers* eigens erstellte Klassen bereitgestellt. Der Quelltext dieses Kapitels befindet sich im Anhang C.

Für die lagerichtige Überlagerung der unterschiedlichen Datenebenen sind verschiedene Bedingungen zu beachten:

Zum einen müssen die Projektionen der einzelnen Datenebenen übereinstimmen. Kommerzielle Kartenvierer wie *Google Maps*, *Live Search Maps* und *Yahoo Maps*, die auch eine offene API anbieten, nutzen die sphärische Mercatorprojektion (vgl. OL Bsp 2008a). In dieser Projektion wird die Erde als Kugel angenommen und nicht, wie üblich, als Ellipsoid. Die Einheit der Koordinaten ist der Meter (vgl. Schmidt C. 2008). Der offizielle EPSG-Code ist 3785. In *OpenLayers* wird jedoch noch der inoffizielle „EPSG-Code“ 900913³³ gebraucht (vgl. OL MailingList 2008a, Schmidt C. 2007). Die Koordinaten können über das Attribut *displayProjection* im WGS 84-Bezugssystem (EPSG-Code: 4326) angezeigt werden. Des Weiteren müssen die maximale Bodenauflösung (*maxResolution*) und Kartenausdehnung (*maxExtent*) festgelegt werden. Die Bodenauflösung ist als Einheit pro Pixel definiert. Die Kartenausdehnung wird durch ein *Bounds*-Objekt festgelegt. Das *Bounds*-Objekt beinhaltet vier Zahlenwerte zur Festlegung eines grenzbeschreibenden Rechtecks. Bei einem Erdumfang von ca. 40 000 000 m ergeben sich die Werte für die linke untere (-20037508.34, -20037508.34) und die rechte obere Ecke (20037508.34, 20037508.34) des Rechtecks.

Die Optionen des Kartenobjektes sehen wie folgt aus:

33 durch Leetspeak (s. Glossar) von dem Wort *google* abgeleitet

```

var options = {
  projection: new OpenLayers.Projection("EPSG:900913"),
  displayProjection: new OpenLayers.Projection("EPSG:4326"),
  maxResolution: 156543.0339,
  maxExtent: new OpenLayers.Bounds(-20037508.34, -20037508.34,
    20037508.34, 20037508.34),
};
var map = new OpenLayers.Map("map", options );

```

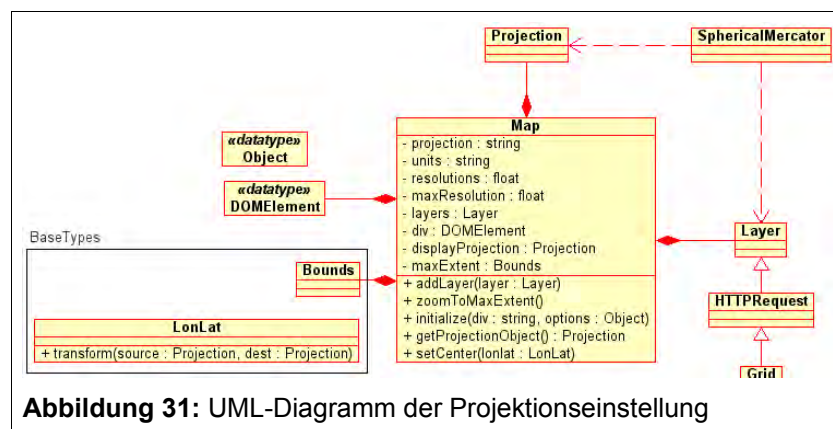
Mit der Funktion *setCenter* wird der Kartenmittelpunkt bestimmt. Um diesen nicht im Format der sphärischen Mercatorprojektion angeben zu müssen, gibt es die Möglichkeit der Punkttransformation. Dafür benötigt man die Ausgangsprojektion und einen Punkt. In diesem Fall ist der Punkt ein *LonLat*-Objekt. Diese werden dann als Argumente der Transformationsfunktion mitgegeben:

```

var proj = new OpenLayers.Projection("EPSG:4326");
var point = new OpenLayers.LonLat(-88, 13);
point.transform(proj, map.getProjectionObject());
map.setCenter(point);

```

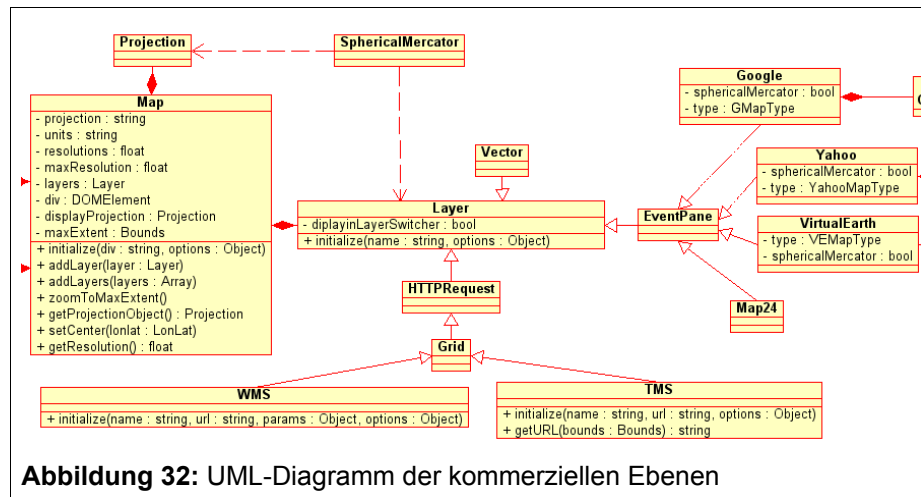
In den einzelnen Ebenen lässt sich die sphärische Mercatorprojektion über *sphericalMercator: true* einstellen (Bsp. s. 5.1). Die Zusammenhänge und Einstellungen fasst das UML-Diagramm in Abbildung 31 zusammen.



Zentrales Element des oberen UML-Diagrammes ist die Klasse *Map*. Die Klasse besitzt Attribute zur Einstellung der Projektion (*projection*) der vorliegenden Daten und der Projektion zum Anzeigen (*displayProjection*) der Daten. Mit der Funktion *getProjection* lässt sich die eingestellte Projektion ermitteln. In diesem Zusammenhang wird die Klasse *Projection* benötigt. Sie ist Datentyp des Attributes *displayProjection* und Rückgabewert der Funktion *getProjection*. Rechts oben im Schaubild ist die Klasse *SphericalMercator* sichtbar. Sie benötigt die beiden Klassen *Projection* und *Layer*. Wie schon zu Beginn des Kapitels erläutert, dient die Klasse *SphericalMercator* der korrekten Darstellung von pro-

prietären Datenquellen.

Die Abbildung 32 veranschaulicht die in diesem Kapitel verwendeten Klassen und Funktionen.



Es sind die vier Klassen *Google*, *Yahoo*, *VirtualEarth* und *Map24* zu sehen, die von der Klasse *EventPane* abgeleitet wurden. Die Klasse *EventPane* (engl. für Ereignis und Glascheibe) wurde wiederum von der Klasse *Layer* abgeleitet. Um die Karten von *Google*, *Yahoo* usw. wie gewohnt mit der Maus navigieren zu können, wurde die *EventPane*-Klasse erstellt. Diese fängt die Mausereignisse ab und leitet sie an die Datenklasse wie zum Beispiel *VirtualEarth* weiter. Außerdem ist in dem obigen Bild die Einordnung der Klasse *TMS*, die genau wie die *WMS*-Klasse von der Klasse *Grid* abgeleitet ist, zu sehen. Die *TMS*-Klasse wird für die Einbindung von *OpenStreetMap*-Daten verwendet (s. Abschnitt 5.5). Auf die *WMS*-Klasse wurde ausführlich im Abschnitt 4.1 eingegangen.

Nach dieser Einleitung in die proprietären Datenquellen können die verschiedenen Ebenen integriert werden.

5.1 Google Maps

Große Aufmerksamkeit haben in den letzten Jahren die Datenebenen von *Google Maps*³⁴ gefunden, die sehr einfach in *OpenLayers* eingebettet werden können. *Google Maps* stellt eine API zur Verfügung. Auf der Homepage von *Google Maps* kann man sich für die API anmelden. Nach der Anmeldung erhält man einen Schlüssel. Über diesen Schlüssel und

³⁴ maps.google.com [25.11.08]

die API erhält man Zugang zu den Daten von *Google Maps*. Die Nutzung der API von *Google Maps* ist zwar kostenlos, aber nur mit bestimmten Einschränkungen gestattet. So muss zum Beispiel die Seite, in die die API eingebunden ist, frei und für jedermann zugänglich sein. Eine lokale Seite darf nur für Entwicklungs- und Testzwecke existieren. Außerdem muss das Logo von *Google* sichtbar sein. *Google* behält sich das Recht vor, Werbung in die Karten zu integrieren.

Mit folgendem Code bindet man die API von *Google Maps* unter Angabe des Schlüssels ein:

```
<script src='http://maps.google.com/maps?file=api&v=2&key=Schlüssel'>
</script>
```

Für *Google Maps* gibt es eine eigene Klasse in *OpenLayers*, die von der Klasse *Layer* abgeleitet wurde. Im Konstruktor werden Name, Kartenart und Projektion festgelegt. Der Name dient der Benennung im *LayerSwitcher*. Sonderzeichen wie zum Beispiel ß und ä müssen im Namen HTML-kodiert angegeben werden, damit sie korrekt dargestellt werden können. Derzeit bietet *Google Maps* vier verschiedenen Kartendarstellungen an. Der Standardkartentyp ist die Straßenkarte, bei *Google Maps* mit `G_NORMAL_MAP` bezeichnet. Außer dieser gibt es noch Satellitenbilder (`G_SATELLITE_MAP`), beschriftete Luftbilder (`G_HYBRID_MAP`) und die Geländekarte (`G_PHYSICAL_MAP`). In Abbildung 33 sind die Vieweransichten der einzelnen Ebenen und der entsprechende Code abgebildet.



	
<pre>var google = new OpenLayers.Layer.Google("Google Stra&szlig;e", {sphericalMercator: true});</pre>	<pre>var google_sat = new OpenLayers.Layer.Google("Google Satellit", {type: G_SATELLITE_MAP, sphericalMercator: true});</pre>

Abbildung 33: Darstellungen von Google Maps in OpenLayers

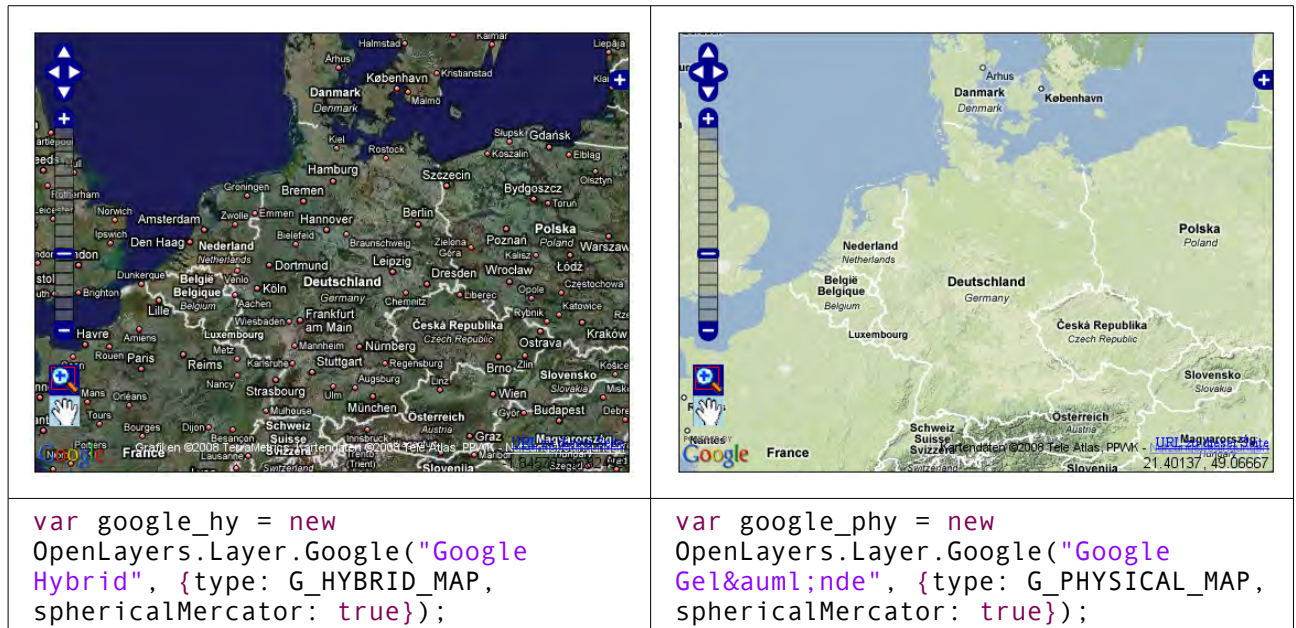


Abbildung 33 (Fortsetzung): Darstellungen von Google Maps in OpenLayers

5.2 Yahoo! Maps

Neben dem Online-Kartendienst *Google Maps* gibt es auch noch *Yahoo! Maps*³⁵. Bei *Yahoo Maps* sind ebenfalls Straßenkarten, Satellitenbilder und hybride Karten verfügbar. Wie diese in *OpenLayers* dargestellt werden können, zeigt Abbildung 34.

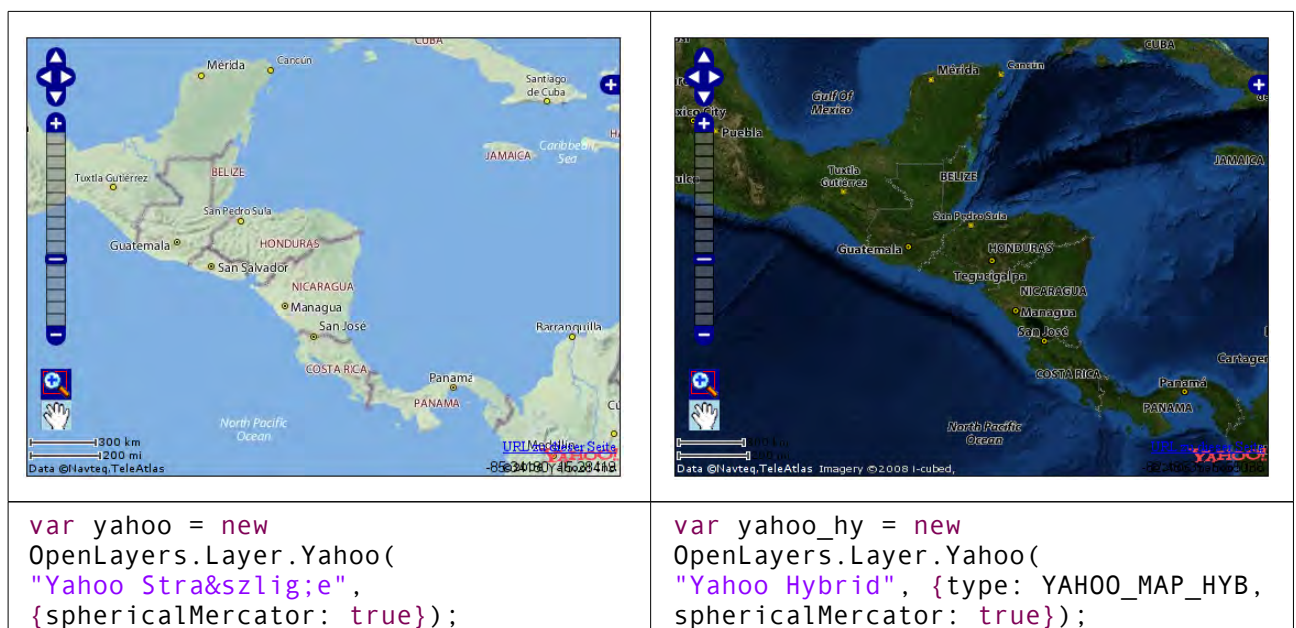


Abbildung 34: Kartenansichten in Yahoo Maps

35 maps.yahoo.com, <http://de.routenplaner.yahoo.com> [25.11.08]

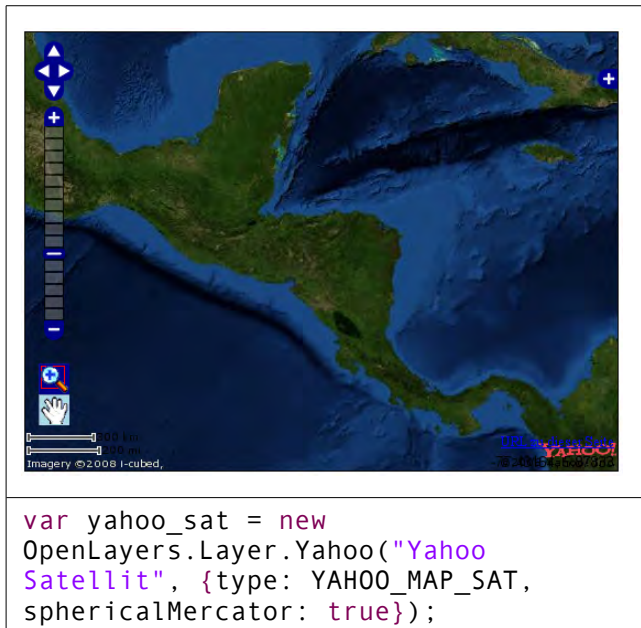


Abbildung 34: Kartenansichten in Yahoo Maps

Nach der Integration der *Yahoo*-Karten tritt allerdings folgendes Problem auf: In der *OpenLayers*-Version 2.7 funktioniert die Zoomfunktion mit der Maus nicht mehr. Man kann nur noch über die Zoomleiste den Maßstab verändern. Wie schon einleitend in diesem Kapitel beschrieben, übernimmt die Klasse *EventPane* die Aufgabe die Mausklicks an die Datenebene weiterzuleiten. Aufgrund von Änderungen in der API von *Yahoo* steht die Funktionalität zur Zeit jedoch nicht zur Verfügung (vgl. OL MailingList 2008b). Das Problem kann durch Hinzufügen einer *WMS*- oder Vektorebene umgangen werden. Diese Ebene wird ausschließlich für die Weiterleitung der Mausklicks benötigt und nicht zur Datenanzeige. Daher soll sie für den Benutzer nicht sichtbar sein und wird im *Layer-Switcher* nicht angezeigt:

```
map.addLayer(new OpenLayers.Layer.Vector(displayInLayerSwitcher: false));
```

5.3 Live Search Maps

*Live Search Maps*³⁶ basiert auf *Microsoft Virtual Earth* und ist mit den Kartendiensten *Google Maps* und *Yahoo Maps* vergleichbar. Auch von Microsoft werden drei Kartenwerke (Straße, Satellit und Hybrid) bereitgestellt. Wie sie in *OpenLayers* eingebunden werden und wie das Ergebnis aussieht, erläutert die Abbildung 35.

³⁶ maps.live.com [25.11.08]




	
<pre>var ve = new OpenLayers.Layer.VirtualEarth("VirtualEarth Stra&szlig;e", {type: VEMapStyle.Road, sphericalMercator: true});</pre>	<pre>var ve_hy = new OpenLayers.Layer.VirtualEarth("VirtualEarth Hybrid", {type: VEMapStyle.Hybrid, sphericalMercator: true});</pre>
	<pre>var ve_luft = new OpenLayers.Layer.VirtualEarth("VirtualEarth Luftbild", {type: VEMapStyle.Aerial, sphericalMercator: true});</pre>

Abb. 35: Kartenwerke von Live Search Maps

5.4 Map24



Schließlich wird *Map24*³⁷ vorgestellt. *Map24* war ursprünglich das Internetportal der deutschen Firma *Mapsolute* und wurde Ende des Jahres 2007 von der Firma *NAVTEQ* aufgekauft, die wiederum von *Nokia* aufgekauft wurde. Der Kartendienst von *Map24* deckt 73 Länder auf sechs Kontinenten ab. Somit steht die Routenplanung in Europa, Amerika,

37 www.de.map24.com [25.11.08]

Afrika, im Nahen Osten und im pazifischen Asien zur Verfügung. *Map24* setzt nicht wie seine Kontrahenten auf Rasterdaten, sondern auf Vektorgrafiken auf. Das hat den Vorteil, dass das Datenvolumen reduziert ist. *Map24* verwendet ein *Java Applet*³⁸, das beim Laden zunächst Zeit in Anspruch nimmt, danach aber eine beachtliche Performance liefert. Neben dem *Java Applet* steht eine Ajax-API für Programmierer zur Verfügung. Diese Schnittstelle kann man in eigene Webseiten einbinden. Den benötigten Schlüssel erhält man unter devnet.map24.com [25.11.08].

```
<script type="text/javascript" language="javascript"
    src="http://api.maptp.map24.com/ajax?apikey=Schlüssel"> </script>
```

Da *Map24* standardmäßig nicht in der *OpenLayers*-Bibliothek enthalten ist, muss die Datei erst von svn.openlayers.org/sandbox/thliese/openlayers/lib/OpenLayers/Layer/Map24.js heruntergeladen und unter *OpenLayers-2.7/lib/OpenLayers/Layer* gespeichert werden (vgl. OL Wiki 2008a). Endgültig inkludiert wird sie durch den Eintrag "[OpenLayers/Layer/Map24.js](#)" in die *OpenLayers.js*-Datei, die sich im *lib*-Ordner befindet.

Eine neue Ebene kann im gleichen Stil wie bei den vorausgegangenen Ebenen erzeugt werden:

```
var map24 = new OpenLayers.Layer.Map24("Map24");
```

Das Ergebnis ist in Abbildung 36 zu sehen. Im Anhang D ist der Sourcecode zu *Map24*.



Abbildung 36: Map24-Ebene in OpenLayers

Leider ist die Implementierung der *Map24*-API in *OpenLayers* nicht vollständig abgeschlossen. Es treten noch einige Fehler auf, zum Beispiel ist die Zoomleiste in umgekehrter Weise zu nutzen. Außerdem ist die Weiterentwicklung und vollständige Integration in das *OpenLayers*-Framework fraglich.

³⁸ Programm, das in Java geschrieben ist und im Webbrowser ausgeführt wird, s. Glossar

5.5 OpenStreetMap

Die Karten der kommerziellen Anbieter sind zwar kostenlos, unterliegen aber verschiedenen Restriktionen. Anders ist das bei *OpenStreetMap* (OSM), hier unterliegen die Daten einer *CC-by-sa-Lizenz*³⁹. Das bedeutet, dass das Kopieren, Vervielfältigen und Verkaufen der darauf aufbauenden Karten erlaubt ist. Dabei sollte man beachten, dass auch das neu entstandene Produkt kopiert werden darf.

Wie bei einem Wiki üblich kann auch bei dem geographischen Wiki *OpenStreetMap* jeder mitarbeiten. „Hobbyisten laufen, wandern, fahren oder reisen sonstwie durch die Gegend und zeichnen dabei die zurückgelegten Wegstrecken mit GPS-Geräten auf“ (Ramm & Topf 2008:3). Diese werden im Nachhinein bearbeitet und in der OSM-Datenbank gespeichert. So entsteht Weg für Weg eine neue Weltkarte.

Bei den Daten von *OpenStreetMap* handelt es sich weder um Raster- noch um Vektordaten. Es werden Geodaten vorgehalten (vgl. Ramm & Topf 2008:6). Somit hat der Nutzer Zugriff auf die Ursprungsdaten und kann mittels Renderprogrammen das Aussehen seiner Karte individuell gestalten. Die beiden wichtigsten Renderer bei OSM sind *Mapnik* und *Osmarender*. *Mapnik* wurde von Artem Pavlenko unabhängig vom OSM-Projekt entwickelt (vgl. Ramm & Topf 2008:149). Die Homepage befindet sich unter www.mapnik.org. *Mapnik* ist in C++ geschrieben und kann für die Web- und Desktopentwicklung genutzt werden. Die OSM-Daten müssen vor dem Rendern konvertiert werden, da *Mapnik* auf einem anderen Datenformat aufbaut. *Mapnik* erzeugt typischerweise Bitmaps im PNG-Format. *Osmarender* wurde von Etienne Cherdlu entwickelt und erzeugt im Gegensatz zu *Mapnik* SVG-Vektorkarten (vgl. Ramm & Topf 2008:127). Das Layout kann individuell angepasst werden. Man kann Farben, Strichstärken und Symbole bestimmen. Ein interessantes Projekt ist *Tiles@home*. Hier werden die Kacheln verteilt auf den Rechnern der Teilnehmer mit *Osmarender* gerendert. Die gerenderten Geodaten werden in Form von Kacheln gespeichert. Wird eine Kachel aufgerufen, die älter als drei Tage ist, wird sie neu erstellt. Kacheln, die vier Wochen alt sind, werden gelöscht (vgl. Ramm & Topf 2008:112ff).

OpenStreetMap präsentiert seine Geodaten mit Hilfe von *OpenLayers*. Der nächste Absatz beschreibt wie man das realisieren kann.

³⁹ Creative Commons-Lizenz: Namensnennung, Weitergabe unter gleichen Bedingungen, s. Glossar

Die Abbildung 37 zeigt die Einbindung zweier OSM-Ebenen in *OpenLayers*. Die erste wurde mit *Mapnik* erstellt und die zweite mit *Osmarender* im Rahmen des *Tiles@home*-Projektes. Hierbei kommt die *TMS*-Ebene zum Einsatz. Dahinter steckt ein Speicherformat für Kacheln (weitere Informationen s. Abschnitt 4.1 III).

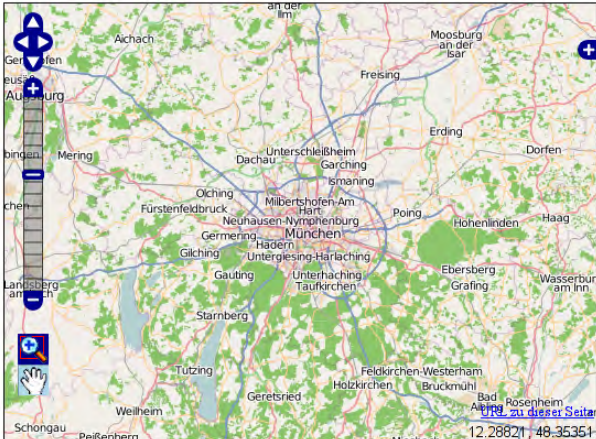
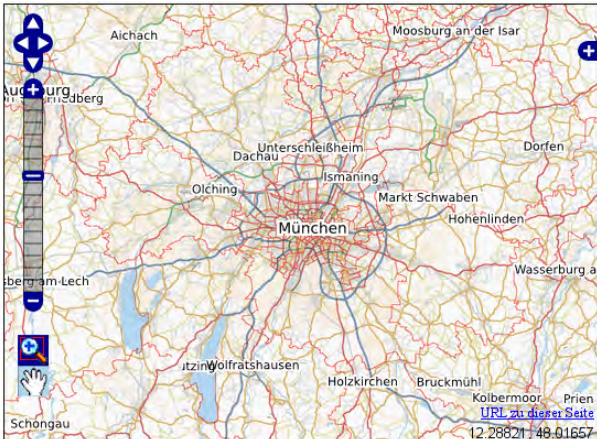
	
<pre>var osm = new OpenLayers.Layer.TMS("OpenStreetMap (Mapnik)", "http://tile.openstreetmap.org/", {type: 'png', getURL: osm_getTileURL, displayOutsideMaxExtent: true});</pre>	<pre>var osmarender = new OpenLayers.Layer.TMS("OpenStreetMap (Tiles@Home)", "http:// tah.openstreetmap.org/Tiles/ tile/", {type: 'png', getURL: osm_getTileURL, displayOutsideMaxExtent: true});</pre>

Abbildung 37: Karten von OpenStreetMap

Für den Zugriff auf die gespeicherten Kacheln wird die *getTileURL*-Funktion benötigt. Diese ermittelt den Pfad zu der gespeicherten Kachel anhand des darzustellenden Kartenausschnittes (*bounds*), des maximalen Kartenausschnittes (*maxExtent*), der Kachelgröße (*tileSize*), der Zoomstufe (*getZoom*) und des Maßstabes (*getResolution*) (vgl. OL Bsp 2008a):

```
function osm_getTileURL(bounds)
{
  var res = this.map.getResolution();
  var x = Math.round(
    (bounds.left - this.maxExtent.left) / (res * this.tileSize.w));
  var y = Math.round(
    (this.maxExtent.top - bounds.top) / (res * this.tileSize.h));
  var z = this.map.getZoom();
  var limit = Math.pow(2, z);

  if (y < 0 || y >= limit)
  {
    return OpenLayers.Util.getImagesLocation() + "404.png"; }
}
```

```

else
{
  x = ((x % limit) + limit) % limit;
  return this.url + z + "/" + x + "/" + y + "." + this.type;
}
}

```

Mit der Funktion *addLayers* des Kartenobjektes kann man die in den vorangegangenen Beispielen definierten Ebenen auf einmal dem Viewer hinzufügen:

```
map.addLayers([google, google_phy, google_hy, google_sat, yahoo, yahoo_hy, yahoo_sat, ve, ve_hy, ve_luft, osm, osmarender, oam]);
```

6 Erfassen, Editieren und Messen

Im sechsten Kapitel geht es um das Erfassen und Verändern von Geometrieobjekten (Quelltext s. Anhang J). Zusätzlich wird die Messfunktion vorgestellt.

JavaScript und HTML sind nicht zum Zeichnen vorgesehen und bieten keine Grafikfähigkeit. Um dennoch Zeichnungen zu visualisieren, hat man sich den *div*-Elementen bedient. Ein *div*-Element definiert einen HTML-Bereich und wird normalerweise zur Formatierung verwendet. Gezeichnet wird nun mit vielen kleinen HTML-Bereichen, die teilweise nur einen Pixel groß sind. Walter Zorn hat dafür die *JavaScript*-Vektorgrafikbibliothek *wz_jsgraphics.js* entwickelt. Sie stellt Funktionen zum Zeichnen von Linien, Kreisen, Ellipsen, Polygonen und Polylinien bereit (vgl. Zorn 2006).

In *OpenLayers* findet man ebenfalls Funktionen, die für das Erfassen von Geometrieobjekten vorgesehen sind. Das Zeichnen findet auf einer eigenen Ebene statt, der Vektorebene *vlayer*. Die *EditingToolbar* liefert die Standardfunktionen für das Zeichnen von Objekten. Mit ihr kann man Punkte, Linien und Polygone zeichnen. Beim Erstellen der Werkzeugleiste wird als Argument die Zeichenebene (*vlayer*) angegeben:

```
var vlayer = new OpenLayers.Layer.Vector("Editieren");
controls:[new OpenLayers.Control.EditingToolbar(vlayer)]
```

Die eingebundene Werkzeugleiste und beispielhafte geometrische Formen sind in Abbildung 38 zu sehen.



Abbildung 38: Digitalisierungsfunktionen in OpenLayers

Die gezeichneten Elemente können auch verschoben werden. Zu diesem Zweck legt man ein *DragFeature*-Steuerelement an. Dabei gibt man wieder die Ebene (*vlayer*) an, auf die das Steuerelement angewendet werden soll. Das Steuerelement wird anschließend in die Werkzeugleiste aufgenommen:

```
var drag = new OpenLayers.Control.DragFeature(vlayer);
var toolb = new OpenLayers.Control.EditingToolbar(vlayer);
toolb.addControls(drag);
```

Über CSS wird das Layout konfiguriert. Zum Beispiel werden dort die Bilder für die Schaltflächen angegeben:

```
.olControlEditingToolbar .olControlDragFeatureItemInactive {
  background-image: url("img/move_feature_off.png");
  background-position: 0px 0px;
  background-repeat: no-repeat;
}
.olControlEditingToolbar .olControlDragFeatureItemActive {
  background-image: url("img/move_feature_on.png");
  background-position: 0px 0px;
  background-repeat: no-repeat;
}
```

Als Ergebnis erscheint in der Werkzeugleiste ein neuer Schalter (s. Abbildung 39). Nach dem Aktivieren verändert sich der Mauszeiger beim Berühren einer Geometrie zu einem Kreuz. Danach kann man die ausgewählte Geometrie mit gedrückter Maustaste verschieben. Das Loslassen der Taste positioniert die Geometrie.



Abbildung 39: Schaltfläche zum Verschieben von Formen

Beispiel: GeoEditor

Der *GeoEditor* von Carsten Eider ist ein Anwendungsbeispiel für die Umsetzung von Erfassungs- und Editiermethoden in *OpenLayers* (vgl. Eider 2008). In dem *GeoEditor* gibt es eine Funktion zum Erfassen von Geometrieobjekten. Außerdem kann man sie nach dem Erfassen editieren, löschen, entlang einer Trennlinie teilen und in eine beliebige Ebene verschieben (s. Abb. 40). Nach der Bearbeitung können die Geometrieobjekte über einen transaktionalen WFS⁴⁰ gespeichert werden.



Abb. 40: Schaltflächen des GeoEditors

6.1 SVG

OpenLayers greift bei der Visualisierung von Objekten nicht auf die Methode der *div*-Elemente zurück. Da diese sehr langsam und umständlich ist, macht sich *OpenLayers* dem SVG-Standard⁴¹ zunutze. Obwohl SVG ein W3C-Standard ist, wird er vom *Internet Explorer* nicht unterstützt. Dieser Browser hält an seinem Format VML⁴² fest. *OpenLayers* stellt automatisch die Daten je nach Browser entweder als VML oder SVG dar (vgl. OL FAQ 2008).

Nun folgt ein Beispiel eines konkreten Polygons sowie dessen Umsetzung im Code und als Grafik (s. Abbildung 41):

```
<svg id="OpenLayers.Layer.Vector_4_svgRoot" width="1248"
height="607" viewBox="0 0 1248 607">
  <g id="OpenLayers.Layer.Vector_4_root"
style="visibility: visible;" transform="">
    <path id="OpenLayers.Geometry.Polygon_204"
d=" M 281,260.5 178,363.5 313,423.5 407,325.5
281,260.5 z" fill="#ee9900"
fill-opacity="0.4" stroke="#ee9900"
stroke-opacity="1" stroke-width="1"
pointer-events="visiblePainted"
cursor="inherit"/>
  </g>
</svg>
```



Abbildung 41:
Darstellung der SVG-
Daten im Browser

Sämtliche SVG-Grafiken befinden sich innerhalb der SVG-Tags `<svg>` und `</svg>`. Dort gibt es die Möglichkeit mit Hilfe des Gruppierungselementes `<g>` und `</g>` Elemente zu

40 Der Transactional Web Feature Service (WFS-T) erweitert den WFS um den schreibenden Zugriff.

41 Vektorgrafikstandard des W3C, s. Glossar

42 Beschreibung von zweidimensionalen Vektorgrafiken, s. Glossar

Einheiten zu verbinden. Hier ist schließlich die eigentliche Geometrie des Polygons eingeordnet. In SVG gibt es Elemente für Linie, Kreis, Ellipse, Rechteck und Polygon. Das Grundelement in SVG ist der Pfad `<path.../>`. Aus ihm können alle anderen Objekte wie Kreis, Linie und Dreieck aufgebaut werden. Hierbei handelt es sich um einen absoluten Pfad, der durch Koordinatenpaare, wie zum Beispiel `281,260.5` beschrieben ist. Da es sich um ein viereckiges Polygon handelt, werden fünf Koordinatenpaare benötigt. Das erste und letzte Paar sind identisch. Dadurch wird das Polygon geschlossen. Anschließend folgen noch die Layouteinstellungen.

Auch die WFS-Polygone des Kapitels 4.2.2 werden mit SVG dargestellt.

6.2 Ausgabe der Geometrie als WKT

Der *Simple Feature Access* ist eine Spezifikation des OGCs und definiert eine allgemein gültige Architektur für geografische Daten und deren Geometrieobjekte. Einen Teil der Spezifikation bildet das *Well-Known-Text-Format* (WKT), das der Repräsentation der Geometrie dient. Das WKT-Format ist im Geoinformatikbereich weit verbreitet, da es leicht verständlich und grundlegend ist.

Ein Punkt mit den Koordinaten 17 als Rechtswert und 3 als Hochwert wird beispielsweise wie folgt dargestellt:

```
Point(17 3)
```

In *OpenLayers* gibt es eine Klasse für WKT. Mit dieser Klasse kann man unter anderem WKT lesen und schreiben. In dem folgenden Beispiel wird beim Berühren einer Geometrie ein Popup erzeugt, das den *Well Known Text* der Geometrie anzeigt. Dazu wird die Methode *write* der Klasse *WKT* verwendet. Der Übergabeparameter der Methode ist das Objekt (*feature*):

```
var wkt = new OpenLayers.Format.WKT();
var options = {hover: true,
  onSelect: function(feature){popup = new
    OpenLayers.Popup.FramedCloud("wkt", // ID
    feature.geometry.getBounds().getCenterLonLat(),
    null, // autosized
    "WKT: " + wkt.write(feature), // contentHTML
    null, // anchor
    false), // closeBox
    map.addPopup(popup)},
  onUnselect: function(feature){popup.destroy()} }
```

```
var select = new OpenLayers.Control.SelectFeature(vlayer, options);
map.addControl(select);
select.activate();
```

Das Ergebnis ist in der Abbildung 42 zu sehen. Dort wurde zunächst ein Dreieck gezeichnet und anschließend mit der Maus das Popup erzeugt. Das Popup zeigt die Koordinaten der drei Punkte. Auch hier benötigt man wieder ein abschließendes Koordinatenpaar, daher sind vier Punkte zu sehen.



6.3 Messen

In Karten stehen zwar in der Regel eine Maßstabsleiste und speziell in *OpenLayers* die Anzeige der Koordinaten zur Verfügung, dennoch ist eine eigene Messfunktion viel komfortabler.

Zunächst einmal bringt jedes Objekt der Klasse *Geometry* eine Funktion mit, die die Länge der Geometrie liefert. Diese Funktion heißt *getLength*. In dem unter 6.2 erzeugten Popup kann zusätzlich zu den Koordinaten im WKT die Länge ausgegeben werden:

```
"<b>WKT:</b> " + wkt.write(feature) + "<br>" +
"<b>Länge:</b> " + feature.geometry.getLength(), // contentHTML
```

Das veränderte Popup zeigt die Abbildung 43:



Will man aber keine neuen Geometrieobjekte erzeugen, sondern nur die Weglänge zwi-

schen zwei Orten messen, so eignet sich eine eigene Messfunktion.

In *OpenLayers* gibt es zum Messen ein eigenes Steuerelement. Mit dem Steuerelement *Measure* kann man Strecken (*Path*) oder Flächen (*Polygon*) messen (vgl. OL Bsp 2008b). Je nachdem gibt man als Parameter *Path* oder *Polygon* an. Ein weiterer Parameter heißt *handlerOptions*. Dort kann man das Layout der Messlinie bzw. -fläche einstellen. Des Weiteren stehen zwei Arten von Ereignissen (*Event-Types*) zur Auswahl: *measure* und *measurepartial*. Mit *measure* zeichnet man eine Linie, schließt mit einem Doppelklick ab und erhält anschließend das Messergebnis. In dem Modus *measurepartial* erhält man schon während der Messung Zwischenergebnisse.

Die Messfunktionalität liefert *OpenLayers*, für die Messergebnisanzeige muss man selbst eine Möglichkeit finden. Hierzu wurde ein *div*-Element mit der ID "lblMessen" dynamisch, also während der Laufzeit erzeugt. Anschließend legt man die Layouteigenschaften des *div*-Elementes fest und hängt es in das DOM⁴³ ein.

Als letztes Element benötigt man einen Schalter. Dafür nutzt man die schon erstellte Klasse *Toggle* (s. 4.2.2). Über diesen Schalter wird die Messfunktion aktiviert und deaktiviert sowie das Anzeigefeld sichtbar und unsichtbar geschaltet.

Der Quelltext gestaltet sich wie folgt (vollständiger Quelltext s. Anhang B):

```
messen = new OpenLayers.Control.Measure(OpenLayers.Handler.Path,
    {handlerOptions: {style: {strokeColor:"#777777", strokeWidth:3},
    persist: true}});
messen.events.on({measurepartial: Messen});

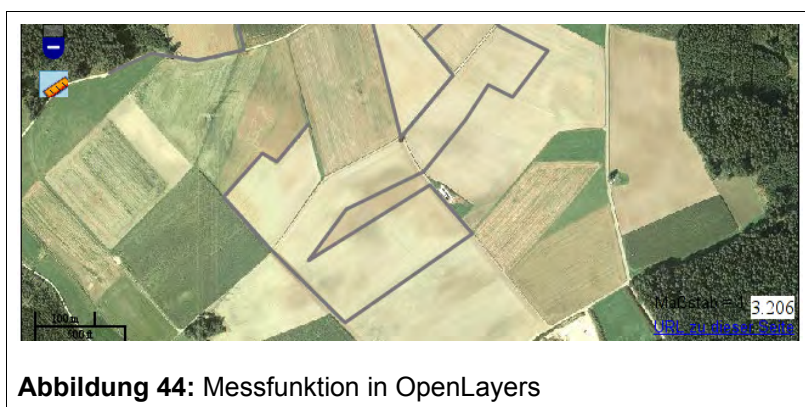
var measure_toggle = new OpenLayers.Control.Toggle({
    displayClass: "mbControlMeasure", title: "Messen"});
var panel = new OpenLayers.Control.Panel({displayClass: "olControlPanel"});
panel.addControls([messen, measure_toggle]);

Label = document.createElement('div');
Label.id = "lblMessen";
Label.style.right = "10px";
Label.style.bottom = "40px";
Label.style.position = "absolute";
Label.style.zIndex = 5001;
Label.style.visibility = "hidden";
Label.style.backgroundColor = "white";
document.body.appendChild(Label);

function Messen(event) {
    Label.innerHTML = event.measure.toFixed(3);
    Label.style.visibility = "visible"; }
```

43 W3C-Spezifikation für den Zugriff auf HTML- und XML-Dokumente, s. Glossar

Links oben in der Abbildung 44 befindet sich die Schaltfläche zum Aktivieren der Messfunktion. Sie zeigt ein kleines Lineal. Wenn man mit dem Mauszeiger darüberfährt, wird *Messen* als Kurzinformation angezeigt. Nachdem die Schaltfläche aktiviert wurde, kann man mit dem ersten Klick die Messlinie beginnen und dann beliebig viele Punkte im Verlauf des Messweges setzen, bis man die Messung durch einen Doppelklick beendet. In der Mitte des Bildes sieht man eine Messlinie, grau dargestellt. Im rechten unteren Bereich ist die Anzeige für die Messergebnisse platziert. Wenn man durch einen Klick auf die Schaltfläche die Messfunktion wieder deaktiviert, verschwinden Messlinie und Anzeigefeld.



7 Passwortschutz

Manche Dienste im Internet sind nicht für jeden zugänglich und werden nur gegen Bezahlung zur Verfügung gestellt. Der Zugang wird zum Beispiel durch ein Passwort geschützt. Eine einfache und übliche Möglichkeit eine Webseite durch ein Passwort zu beschränken ist die HTTP-Authentifizierung. Der Benutzer muss sich beim Webserver über Eingabe von Benutzername und Passwort als zugriffsberechtigt ausweisen. Der Server überprüft anschließend die Angaben und gestattet dem Nutzer den Zugang, falls er dazu berechtigt ist. Es gibt mehrere Verfahren der HTTP-Authentifizierung, die *Basic*- und *Digest-Access*-Authentifizierung. Die häufigste Art ist die *Basic*-Authentifizierung. Dort werden die Authentisierungsdaten kodiert verschickt.

Wenn eine passwortgeschützte Seite aufgerufen wird, so kommt vom Browser eine automatische Passwortabfrage in Form eines Popups. Im Falle von *OpenLayers*, wo in der Regel viele Kacheln einzeln angefragt werden, führt dieser Ansatz zu unzähligen Dialogen

7.3 Passwort im Proxyskript

Die bisher genannten Lösungsvorschläge sind nur für domäneneigene Zugriffe gültig. Die Authentisierungsdaten für eine andere Domäne in den Client *OpenLayers* zu integrieren ist weitaus komplizierter.

Zunächst muss man sich wieder dem bereits im Abschnitt 4.2 beschriebenen Trick des Proxyskriptes (s. Anhang F) bedienen, um mit Ajax auf andere Domänen zugreifen zu können. Dazu wird der entsprechende Server unter *allowedHosts* im Proxyskript *proxy.cgi* eingetragen. Zusätzlich wird das vorhandene Skript für die Speicherung der Authentisierungsdaten genutzt.

Es gibt mehrere Methoden für HTTP-Anfragen, zum Beispiel GET, POST, HEAD oder PUT. Die einzelnen Kacheln werden über die GET-Methode vom WMS-Server angefordert. In der GET-Anfrage sollen auch Benutzerkennung und Passwort übertragen werden.

Das Proxyskript ist in der Programmiersprache *Python*⁴⁵ geschrieben. Im Skript gibt es eine Verzweigung für POST- und GET-Anfragen. In dem Alternativ-Zweig (*else*) wird die Authentisierungsinformation in den HTTP-Kopf der GET-Anfragen geschrieben. Zum Öffnen von URLs nutzt man die Bibliothek *urllib2*. Mit der Funktion *Request* erstellt man eine URL-Anfrage mit den Pflichtparametern *url*, *data* und *headers*. In *url* ist eine Zeichenkette einer gültigen URL zu übergeben. *data* ist für zusätzliche Daten vorgesehen, muss allerdings in diesem Fall mit *None* belegt werden. Würde man den *data*-Parameter nutzen, so wäre die HTTP-Anfrage kein GET mehr, sondern eine POST-Methode. Da aber zum Anfragen der Kacheln eine GET-Methode benötigt wird, setzt man den Parameter *data* auf *None*. Der dritte Parameter *headers* enthält schließlich die für die Authentisierung relevanten Daten. Mit der *urlopen*-Funktion wird die URL geöffnet (vgl. Python 2008):

```
if method == "POST":
    ...
else:
    headers = {"Authorization": "Basic c2NobWl0dDp1cmxhdWI="}
    r = urllib2.Request(url, None, headers)
    y = urllib2.urlopen(r)
```

Bei der Ebenendefinition für den Zugang zum passwortgeschützten WMS gibt man eine URL an, die auf das Proxyskript zeigt. Dieser URL wird im *Query* eine weitere URL als Parameter angehängt. Die URL, die als Parameter mitgeschickt wird, muss kodiert sein. Da

⁴⁵ www.python.org [11.12.08]

in einer URL bestimmte Zeichen reserviert sind, müssen diese im Parameter kodiert angegeben werden (Prozentkodierung). Folgend ist eine **normale** URL und eine **kodierte** URL angegeben. Die reservierten Zeichen sind **fett** dargestellt.

```
http://geodaten.bvv.bybn.de/ogc/ogc_dipl_auth.cgi?
http%3A%2F%2Fgeodaten%2Ebv%2Ebybn%2Ede%2Fogc%2Fogc%5Fd%5Fdipl%5Fauth%2Ecgi%3F
```

Die kodierte URL wird beim Erzeugen des *WMS*-Layers angegeben:

```
wms_auth = new OpenLayers.Layer.WMS("WMS Auth",
                                     "/cgi-bin/proxy.cgi?",
                                     url=http%3A%2F%2Fgeodaten%2Ebv%2Ebybn%2Ede%2Fogc%2Fogc%5Fd%5Fdipl%5Fauth%2Ecgi%3F",
                                     {layers: 'tk', format: 'image/png', srs:'EPSG:31468', version:'1.1.1'})...
```

Die weiteren Parameter für den WMS, die ebenfalls Teil der URL sind, werden während der Laufzeit erzeugt. Um sicherzustellen, dass die WMS-Parameter richtig übergeben werden, müssen im Quelltext weitere Veränderungen vorgenommen werden.

In der Klasse *HTTPRequest* wurde die Funktion *getFullRequestString* abgeändert. Es wurde eine Verzweigung (*if*) eingebaut, um den normalen Programmablauf sicherzustellen. Die *if*-Anweisung unterscheidet zwischen dem normalen und passwortgeschützten (*authentication*) Zugriff. Normalerweise wird die Parameter-Zeichenkette bestimmt. Im neuen Zweig wird die Zeichenkette zusätzlich mit der Funktion *encodeURIComponent* kodiert:

```
if (this.authentication)
    {paramsString = encodeURIComponent(
                                     OpenLayers.Util.getParameterString(allParams));}
else {paramsString = OpenLayers.Util.getParameterString(allParams);}
```

Anschließend wird die Parameter-Zeichenkette mit der URL von der Ebenendefinition (*requestString*) zusammengefügt. Vorher wird jedoch überprüft, ob die *url* mit einem Frage- oder Und-Zeichen endet. Da in dem neu eingefügten Authentisierungsfall das Fragezeichen bereits kodiert ist, muss das kodierte Fragezeichen (*%3F*) in die Überprüfung mit aufgenommen werden. Dazu werden die letzten drei Zeichen der *url* der Variable *lastServerCharencoded* zugewiesen:

```
var lastServerCharencoded = url.slice(url.length - 3);
if ((lastServerChar == "&") || (lastServerChar == "?")
    || (lastServerCharencoded == "%3F"))
    {requestString += paramsString;
```

Das Attribut *authentication* zur Differenzierung des Authentisierungsfalls vom Normalfall wurde als neues Attribut vom Typ *Boolean* in der Klasse *Layer* eingeführt. Standard-

mäßig ist das Attribut auf *false* gesetzt:

```
OpenLayers.Layer = OpenLayers.Class({... authentication: false,
```

Wenn eine Ebene für den Zugang zu einem passwortgeschützten WMS erstellt wird, muss das Attribut *authentication* auf *true* gesetzt werden:

```
wms_auth = new OpenLayers.Layer.WMS( "WMS Auth", "/cgi-bin/proxy.cgi?  
url=http%3A%2F%2Fgeodaten%2Ebv%2Eby%2Ede%2Fogc%2Fogc%5Fdipl%5Fauth%2Ecgi%3F",  
    {layers: 'tk', format: 'image/png', srs: 'EPSG:31468', version: '1.1.1'},  
    {authentication: true});
```

Damit ist die Implementierung der Authentisierungsfunktion in den Client *OpenLayers* abgeschlossen.

Von *OpenLayers* ist der Zugriff auf passwortgeschützte Datenquellen nicht vorgesehen. Die Implementierung einer geschützten Quelle ist zwar möglich, aber nur mit mehreren Änderungen direkt im *OpenLayers*-Quellcode. Bei einem Update von *OpenLayers* auf eine höhere Version, müsste der spezifische Code zusätzlich integriert werden. Außerdem ist darauf hinzuweisen, dass in diesem Kapitel lediglich die einfachste Authentisierungsmethode realisiert wurde. Mit ihr werden die sensiblen Daten zwar kodiert, aber nicht verschlüsselt übertragen. Das bedeutet, dass die Anmeldedaten abhörbar sind.

Mit *Hypertext Transfer Protocol Secure* (HTTPS) wird eine verschlüsselte Verbindung aufgebaut, die nicht abhörbar ist. Diese Option zu integrieren wäre mit weitaus höherem Aufwand verbunden.

8 Schluss

Im Rahmen der Diplomarbeit wurden die Funktionen von *OpenLayers* untersucht. Besonderes Augenmerk wurde auf die Einbindung von *Web-Map-Diensten* und die Performance-Steigerung bei der Einbindung der Daten gerichtet (s. Abschnitt 4.1). Dazu wurden mehrere Möglichkeiten aufgezeigt und untersucht. Einen Kartenausschnitt gekachelt anzufordern, bietet zwar Vorteile für eine dynamische Kartendarstellung, erhöht allerdings auch die Last auf der Serverseite. Diese Last wird durch den Einsatz eines *Tile-Caches* verringert (s. Abschnitt 4.1 III).

Außerdem wurde auch ein direkter Zugriff auf die Kartendaten realisiert (s. Abschnitt 4.1 IV). Hierbei wird ohne WMS und *TileCache* direkt auf die Rasterdateien zugegriffen, was gegenüber dem *TileCache* erneut einen Performance-Vorteil bringt.

Nicht nur die Anzeige eines *Web Map Service* (WMS) ist mit *OpenLayers* möglich, sondern auch die Einbindung eines *Web Feature Service* (WFS). Es werden Punkt- und Polygonobjekte eines WFS in *OpenLayers* visualisiert und die zugehörigen Sachinformationen in Form eines Popup-Fensters dargestellt (s. Abschnitt 4.2).

Die Einbindung eines *GeoRSS-Feed* in *OpenLayers* wurde ebenfalls gezeigt (s. Abschnitt 4.3). Dadurch können über einen Abonnement-Dienst georeferenzierte Daten auf einer Karte sichtbar gemacht werden.

Neben den standardkonformen Datenquellen, wie zum Beispiel WMS und WFS, wurden auch proprietäre Quellen wie *Google Maps*, *Yahoo Maps* und *Map24* in den Viewer integriert (s. Kapitel 5).

Weiterhin wurden die Grafikfunktionen von *OpenLayers* untersucht. Das Erfassen von Punkten, Linien, Polylinien und Polygonen wurde integriert und getestet (s. Kapitel 6).

Im Rahmen der Arbeit wurde exemplarisch eine Messfunktion mit Hilfe von *OpenLayers* implementiert. Mit dieser Funktion ist es möglich, Strecken und Flächen zu messen (s. Abschnitt 6.3).

Auch die Zugriffsmöglichkeiten auf passwortgeschützte Dienste wurden beleuchtet. Dabei musste in den Quellcode von *OpenLayers* eingegriffen werden, da *OpenLayers* den

Zugriff auf passwortgeschützte Dienste nicht vorsieht. Die Authentisierungsdaten wurden dazu in den *OpenLayers*-Client eingebettet (s. 7. Kapitel).

Das große Spektrum an Steuerelementen wurde untersucht und für verschiedene Testanwendungen verwirklicht (s. Kapitel 3). Besonders intensiv wurde der *LayerSwitcher* als zentrales Element betrachtet. Dieser wurde optisch und funktionell an bestimmte Anwendungen angepasst. Auch die Erstellung eines neuen Steuerelementes wurde gezeigt.

Mit Hilfe von *OpenLayers* kann man das *Well-Known-Text*-Format lesen und schreiben. Diese Funktion wurde ebenfalls beschrieben (s. Abschnitt 6.2).

Im Rahmen der Einbindung der Rasterkartenwerke wurde auch auf die Problematik eingegangen, dass der Kartenmaßstab nur durch vorher getroffene Annahmen bezüglich des Ausgabegerätes bestimmt werden kann (s. Abschnitt 3.6).

Die Bandbreite der Funktionen, die *OpenLayers* zur Verfügung stellt, ist beeindruckend und die Leistung des Frameworks beachtlich. Besonders bemerkenswert sind die umfangreichen Optionen Datenquellen einzubinden. Nicht nur die Integrationsmöglichkeiten von Datenebenen sind vielfältig, sondern auch das Angebot an Steuerelementen und unterstützten Datenformaten. *OpenLayers* bietet umfassende Möglichkeiten zur Erstellung eines Viewers. *OpenLayers* unterstützt dabei nicht nur Standards, sondern auch proprietäre Spezialfälle.

Bemerkenswert ist die SVG-Technologie, die *OpenLayers* zur Visualisierung der Geometrieobjekte verwendet. Dadurch werden die Darstellung komplexer Grafiken und die Transparenz von Objekten ermöglicht. Der W3C-Standard wird von allen Webbrowsern unterstützt, mit Ausnahme des Internet Explorers von Microsoft. Dieser Browser unterstützt im Bereich der Vektordarstellung ausschließlich sein eigenes Format VML. Dennoch können Vektorgrafiken problemlos von *OpenLayers* im Internet Explorer durch die ebenfalls implementierte VML-Unterstützung angezeigt werden. *OpenLayers* erweitert damit die Möglichkeiten Geodaten browserbasiert darzustellen. Somit können neben Rasterdaten auch sehr einfach Vektordaten dargestellt werden.

Mit *OpenLayers* ist es sehr leicht möglich, einen browserbasierten Kartenviewer zu erstellen. Der Einstieg ist dabei noch relativ einfach, aber mit wachsenden Anforderungen an

die zu erstellende Anwendung nimmt auch die Zeit der Einarbeitung zu, hauptsächlich aufgrund der unvollständigen Dokumentation. Dennoch gibt es gute Hilfestellungen durch die Beispiele anderer *OpenLayers*-Nutzer und durch die *MailingList*.

Der Anwender muss sich nicht unbedingt mit den internen Programmstrukturen auseinandersetzen, will man jedoch tiefer einsteigen, so offenbart sich der große Umfang und die interne Komplexität des Frameworks. Der Programmcode ist zwar objektorientiert aufgebaut, aber *JavaScript* unterstützt keine echte Objektorientierung. Deshalb bleiben Zusammenhänge zwischen „Klassen“, „Attributen“ und „Methoden“ oft unklar.

Eine gute Unterstützung im Bereich der Webentwicklung erhält man durch die *Firefox*-Erweiterung *Firebug*. Der *Firebug* macht es möglich, die Ladevorgänge der Dateien zu überwachen, Befehle während der Laufzeit des Programmes einzugeben, Einblick in das *Document Object Model* (DOM) zu erhalten und vieles andere mehr.

Die Anforderungen des Auftraggebers, des *Bayerischen Landesamtes für Vermessung und Geoinformation* (LVG), waren die *Web-Map-Dienste* der Bayerischen Vermessungsverwaltung einzubinden, Caching und Tiling zur Performance-Steigerung einzusetzen, Daten ohne WMS direkt aus dem Dateisystem zu laden, auf passwortgeschützte *Web Map Services* zuzugreifen und Daten eines WFS einzubinden. Alle diese Aspekte wurden untersucht und implementiert. Darüber hinaus wurden viele weitere Funktionen von *OpenLayers* behandelt.

Diese Arbeit kann als Grundlage für weitere Überlegungen dienen, einen Kartenviewer mit Hilfe von *OpenLayers* umzusetzen und das Erscheinungsbild von *OpenLayers* nach eigenen Richtlinien anzupassen. Durch den modularen Aufbau von *OpenLayers* ist es möglich, die Oberfläche dementsprechend umzugestalten.

Durch *OpenLayers* ergeben sich vielfältige neue Chancen und Möglichkeiten im Geoinformationsbereich, wovon Datenanbieter und Datennutzer profitieren können. So können Datenanbieter leichter einen Zugang zu Geodaten schaffen und einen Viewer mit wenig Aufwand gleich dazuliefern. Datennutzer können leicht externe Datendienste einbinden und so externe Basiskartenwerke und Fachdaten kombinieren.

Ein Versorgungsunternehmen könnte zum Beispiel als Datenanbieter einfacher seine Leitungsnetze mit einem Viewer bereitstellen. Das hätte mehrere Vorteile. Intern (Intra-

net) könnten mit beschränktem Aufwand leistungsfähige, browserbasierte Erfassungs- und Auskunftskomponenten bereitgestellt werden. Dort könnten beliebige, fachliche und ortsbezogene Details (*Points of Interest*) erfasst, dokumentiert und mit Fotos belegt, dargestellt werden. Extern (Internet) kann dann, quasi als Nebenprodukt, eine - gegebenenfalls im Detaillierungsgrad reduzierte - Ansicht im Browser angeboten werden. Ähnliche Vorteile hätten beispielsweise auch Kommunen in Bezug auf Tourismus, das Amt für Denkmalschutz bei der Darstellung von Denkmälern oder die Immobilienwirtschaft, um nur einige zu nennen.

Die bisher verwirklichten Funktionen von *OpenLayers* sind enorm vielfältig, leistungsfähig und vielversprechend. Daher wird es sich lohnen, das Projekt weiterzuverfolgen und sich selbst in die Gemeinschaft der Entwickler und Anwender einzubringen.

Anhang

Anhang A: UML-Diagramm

Anhang B: WMS, TileCache, mehrere URLs, Messen

s. Abschnitte 4.1 WMS, 4.1 III. TileCache, 4.1 V. WMS-Zugriff mit mehreren URLs, 6.3 Messen

```
// Erstellen des Messsteuerlementes

messen = new OpenLayers.Control.Measure(OpenLayers.Handler.Path,
    {handlerOptions: {style: {strokeColor:"#777777", strokeWidth:3,
        fillColor:"#ffff00", fillOpacity:.5}, persist: true}});
messen.events.on({measurepartial: Messen});

// Erstellen des Schalters für die Messfunktion

var measure_toggle = new OpenLayers.Control.Toggle(
    {displayClass: "mbControlMeasure", title: "Messen"});
var panel = new OpenLayers.Control.Panel({displayClass: "olControlPanel"});
panel.addControls([measure_toggle]);

Label = document.createElement('div');
Label.id = "lblMessen";
Label.style.right = "10px";
Label.style.bottom = "40px";
Label.style.position = "absolute";
Label.style.zIndex = 5001;
Label.style.visibility = "hidden";
Label.style.backgroundColor = "white";
document.body.appendChild(Label);

function Messen(event) {
    Label.innerHTML = event.measure.toFixed(3);
    Label.style.visibility = "visible";
}

var ls = new OpenLayers.Control.LayerSwitcher();

var options = {
    resolutions: [150,100,40,12.5,10,4,2.1166667],
    maxExtent: new OpenLayers.Bounds(4281000, 5235000, 4640000, 5606000),
    units: 'm', // Einheit der Karte
    projection: "EPSG:31468",
    tileSize: new OpenLayers.Size(500,500),
// Kachelgröße in Pixeln, Standard sind 256

    controls:[
        new OpenLayers.Control.PanZoomBar(),
        new OpenLayers.Control.Navigation(),
        new OpenLayers.Control.MousePosition(),
        new OpenLayers.Control.Attribution(),
        new OpenLayers.Control.ScaleLine(),
        new OpenLayers.Control.Scale(),
        new OpenLayers.Control.Permalink(),
        ls,
        messen
    ]
};
```

```

// Kartenobjekt mit HTML-ID.

    var map = new OpenLayers.Map( "map", options );
    ls.maximizeControl();

// Erstellen der Ebenen

var uk = new OpenLayers.Layer.WMS("UK 500",
    "http://www.geodaten.bayern.de/ogc/getogc.cgi?" ,
    {layers: 'uk500', format: 'image/png'});

var dop = new OpenLayers.Layer.WMS("DOP" ,
    "http://www.geodaten.bayern.de/ogc/getogc.cgi?" ,
    {layers: 'dop', format: 'image/jpeg', version:'1.1.1' });

var tk_old = new OpenLayers.Layer.WMS("TK old",
    "http://geodaten.bvv.bybn.de/ogc/ogc_dipl.cgi?",
    {layers: 'tk', format: 'image/png'},
    {singleTile: true, ratio: 1,
    attribution: "(c) Bayerische Vermessungsverwaltung});

var tk = new OpenLayers.Layer.WMS("TK",
    "http://geodaten.bvv.bybn.de/ogc/ogc_dipl.cgi?",
    {layers: 'tk', format: 'image/png'});

var tk_url = new OpenLayers.Layer.WMS("TK URLs",
    ["http://geodaten.bvv.bybn.de/ogc/ogc_dipl.cgi?",
    "http://url1/ogc/ogc_dipl.cgi?",
    "http://url2/ogc/ogc_dipl.cgi?",
    "http://url3/ogc/ogc_dipl.cgi?",
    "http://url4/ogc/ogc_dipl.cgi?"],
    {layers: 'tk', format: 'image/png'});

tc = new OpenLayers.Layer.WMS("TileCache",
    "http://localhost/tilecache-2.04/tilecache.cgi",
    {layers: 'tk', format: 'image/png', version:'1.1.1'});

tc2 = new OpenLayers.Layer.TileCache("Layer.TileCache",
    "http://localhost/tilecache",
    "tk",
    {format: 'image/png'});

tms = new OpenLayers.Layer.TMS("TMS",
    "http://localhost/tilecache-2.04/tilecache.cgi/",
    {type: 'png', layername: 'tk'});

// Hinzufügen der Layer zu der Karte

    map.addLayers([uk, tk_old, tk, dop, tc, tc2, tms, tk_url]);
    map.addControl(panel);

var RW = 4468442;
var HW = 5336771;
map.setCenter(new OpenLayers.LonLat(RW, HW));

```

```

/*
// Anzeigen der Koordinaten durch Mausklick
OpenLayers.Control.Click = OpenLayers.Class(OpenLayers.Control, {
    defaultHandlerOptions: {
        'single': true,
        'double': false,
        'pixelTolerance': 0,
        'stopSingle': false,
        'stopDouble': false
    },
    initialize: function(options) {
        this.handlerOptions = OpenLayers.Util.extend(
            {}, this.defaultHandlerOptions
        );
        OpenLayers.Control.prototype.initialize.apply(
            this, arguments
        );
        this.handler = new OpenLayers.Handler.Click(
            this, {
                'click': this.trigger
            }, this.handlerOptions
        );
    },

    trigger: function(e) {
        var lonlat = map.getLonLatFromViewPortPx(e.xy);
        alert("Gauss-Krueger-Koordinaten:\n\nRW: " + lonlat.lon + " HW: " +
lonlat.lat);
    }

});

var click = new OpenLayers.Control.Click();
map.addControl(click);
click.activate();*/

```

Anhang C: Proprietäre Geodaten

s. Kapitel 6

HTML-Datei

```
<html>
<head>
<title>OpenLayers: diverse Layer</title>
<!-- this gmaps key generated for http://openlayers.org/dev/ -->

<script src="http://maps.google.com/maps?
file=api&v=2&key=ABQIAAAjpkAC9ePGem0lIq5XcMiuhr_wLWPFku8Ix9i2SXYRVK3e45
q1BQUd_beF8dtzKET_EteAjPdGDwqpQ"></script>
<script src="http://api.maps.yahoo.com/ajaxymap?v=3.0&appid=euzuro-openlayers">
</script>

<script src="http://dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=6.1">
</script>

<script type="text/javascript" src="OpenLayers-2.7/lib/OpenLayers.js">
// Einbindung der OpenLayers-Bibliothek
</script>

<style type="text/css">
  #map {
    width: 100%;
    height: 100%;
    border: 1px solid black;
  }
</style>
</head>
<body>
  <div id='map'>
  </div>
  <script type="text/javascript" src="js/OpenLayers_Google.js">
  </script>
</body>
</html>
```

JavaScript-Datei

```
// Karteneinstellungen, einschliesslich der Steuerelemente

var ls = new OpenLayers.Control.LayerSwitcher();
var options = {
  projection: new OpenLayers.Projection("EPSG:900913"),
  displayProjection: new OpenLayers.Projection("EPSG:4326"),
  maxResolution: 156543.0339,
  maxExtent: new OpenLayers.Bounds(-20037508.34, -20037508.34, 20037508.34,
20037508.34),
  controls: [
    ls,
    new OpenLayers.Control.PanZoomBar(),
    new OpenLayers.Control.Permalink('permalink'),
    new OpenLayers.Control.MousePosition(),
    new OpenLayers.Control.MouseToolbar()
  ]
};

// getTileURL-Funktion für TMS-Ebenen von OSM

function osm_getTileURL(bounds) {
  var res = this.map.getResolution();
  var x = Math.round((bounds.left - this.maxExtent.left) / (res *
this.tileSize.w));
  var y = Math.round((this.maxExtent.top - bounds.top) / (res *
this.tileSize.h));
  var z = this.map.getZoom();
  var limit = Math.pow(2, z);

  if (y < 0 || y >= limit) {
    return OpenLayers.Util.getImageLocation() + "404.png";
  } else {
    x = ((x % limit) + limit) % limit;
    return this.url + z + "/" + x + "/" + y + "." + this.type;
  }
}

// OpenLayers.Map Konstruktor benötigt ein Argument, dieses kann die ID eines
HTML-Elementes sein.

// In dieses Element wird die Karte platziert.

var map = new OpenLayers.Map( "map", options );

// Erstellen der Ebenen mittels der Layer-Konstruktoren

var google = new OpenLayers.Layer.Google("Google Stra&szlig;e",
  {sphericalMercator: true});
var google_phy = new OpenLayers.Layer.Google("Google Gel&auml;nde",
  {type: G_PHYSICAL_MAP, sphericalMercator: true});
var google_hy = new OpenLayers.Layer.Google("Google Hybrid",
  {type: G_HYBRID_MAP, sphericalMercator: true});
var google_sat = new OpenLayers.Layer.Google("Google Satellit",
  {type: G_SATELLITE_MAP, sphericalMercator: true});

var yahoo = new OpenLayers.Layer.Yahoo("Yahoo Stra&szlig;e",
  {sphericalMercator: true});
```

```

var yahoo_hy = new OpenLayers.Layer.Yahoo("Yahoo Hybrid",
    {type: YAHOO_MAP_HYB, sphericalMercator: true});
var yahoo_sat = new OpenLayers.Layer.Yahoo("Yahoo Satellit",
    {type: YAHOO_MAP_SAT, sphericalMercator: true});

var ve = new OpenLayers.Layer.VirtualEarth("VirtualEarth Stra&szlig;e",
    {type: VEMapStyle.Road, sphericalMercator: true});
var ve_hy = new OpenLayers.Layer.VirtualEarth("VirtualEarth Hybrid",
    {type: VEMapStyle.Hybrid, sphericalMercator: true});
var ve_luft = new OpenLayers.Layer.VirtualEarth("VirtualEarth Luftbild",
    {type: VEMapStyle.Aerial, sphericalMercator: true});

var osm = new OpenLayers.Layer.TMS("OpenStreetMap (Mapnik)",
    "http://tile.openstreetmap.org/",
    {type: 'png', getURL: osm_getTileURL,
    displayOutsideMaxExtent: true});

var oam = new OpenLayers.Layer.TMS("OpenAerialMap",
    "http://tile.openaerialmap.org/tiles/1.0.0/openaerialmap-900913/",
    {type: 'png', getURL: osm_getTileURL});

var osmarender = new OpenLayers.Layer.TMS("OpenStreetMap (Tiles@Home)",
    "http://tah.openstreetmap.org/Tiles/tile/",
    {type: 'png', getURL: osm_getTileURL,
    displayOutsideMaxExtent: true});

// Hinzufügen der Layer zu der Karte

map.addLayers([google, google_phy, google_hy, google_sat, yahoo, yahoo_hy,
    yahoo_sat, ve, ve_hy, ve_luft, osm, osmarender, oam]);
map.addLayer(new OpenLayers.Layer.Vector({},
    {displayInLayerSwitcher: false}));

ls.maximizeControl();

// Kartenmittelpunkt festlegen

var proj = new OpenLayers.Projection("EPSG:4326");
var point = new OpenLayers.LonLat(-88, 13);
point.transform(proj, map.getProjectionObject());
map.setCenter(point,2);

```


Anhang D: Map24

s. Abschnitt 5.4.

HTML-Datei

```
<html>
<head>
<title>OpenLayers: Map24</title>
<!-- this map24 key generated for
      http://dev.openlayers.org/sandbox/thliese/openlayers/examples/ -->
<script type="text/javascript" language="javascript"
src="http://api.maptp.map24.com/ajax?
      appkey=FJX8b3c1e9dfb7870b0e74d042784ab7X35"> </script>

<script type="text/javascript" src="OpenLayers-2.7/lib/OpenLayers.js">
// Einbindung der OpenLayers-Bibliothek
</script>

<style type="text/css">
  #map {
    width: 100%;
    height: 100%;
    border: 1px solid black;
  }
</style>
</head>
<body>
  <div id='map'>
  </div>
  <script type="text/javascript" src="js/OpenLayers_Map24.js">
  </script>
</body>
</html>
```

JavaScript-Datei

```
// Karteneinstellungen

var options = {
  controls: [
    new OpenLayers.Control.PanZoomBar(),
    new OpenLayers.Control.Permalink('permalink'),
    new OpenLayers.Control.MousePosition(),
    new OpenLayers.Control.MouseToolbar()
  ]};

// Kartenobjekt
var map = new OpenLayers.Map( "map", options );

// Layer erstellen

var map24 = new OpenLayers.Layer.Map24("Map24");

// Hinzufügen des Layers zur Karte
map.addLayer(map24);
```

Anhang E: Direkter Zugriff

s. Textabschnitt 4.1 IV.

```
function get_url (bounds) { // notwendig für TMS
    var res = this.map.getResolution();
    var x = bounds.left;
    var y = bounds.bottom;
    var z = res*100;

    var path = z + "/" + x + "/" + x + y + "." + this.type;
    var url = this.url;
    if (url instanceof Array) {
        url = this.selectUrl(path, url);
    }
    return url + path;
}

var options = { // von Map

    resolutions: [40,16,8,4],
    maxExtent: new OpenLayers.Bounds(4280000, 5200000, 4680000, 5640000),
    units: 'm',
    projection: "EPSG:31468",
    tileSize: new OpenLayers.Size(500,500),
    controls:[
        new OpenLayers.Control.PanZoomBar(),
        new OpenLayers.Control.Permalink('permalink'),
        new OpenLayers.Control.MousePosition(),
        new OpenLayers.Control.ScaleLine(),
        new OpenLayers.Control.MouseToolbar(),
    ]
};

// In dieses Element wird die Karte platziert

var map = new OpenLayers.Map("map", options);

// Erstellen der Ebenen

tms_uk500 = new OpenLayers.Layer.TMS("TMS UK500",
    "http://localhost/uk500/", {type: 'png', getURL: get_url});
tms_tk50_16 = new OpenLayers.Layer.TMS("TMS TK50 16",
    "http://localhost/tk50/", {type: 'gif', getURL: get_url});
tms_tk50_8 = new OpenLayers.Layer.TMS("TMS TK50 8",
    "http://localhost/tk50/", {type: 'gif', getURL: get_url});
tms_tk50_4 = new OpenLayers.Layer.TMS("TMS TK50 4",
    "http://localhost/tk50/", {type: 'gif', getURL: get_url});
```

```
// Hinzufügen der Layer zu der Karte

map.addLayers([tms_uk500,tms_tk50_16,tms_tk50_8, tms_tk50_4]);

var RW = 4468442;
var HW = 5336771;
map.setCenter(new OpenLayers.LonLat(RW, HW));

// Ein- bzw. Ausblenden der Ebenen je nach Zoomstufe

map.events.register("zoomend", map, function() {
  switch(map.getResolution()) {
  case 40:
    map.setBaseLayer(tms_uk500);
    break;
  case 16:
    map.setBaseLayer(tms_tk50_16);
    break;
  case 8:
    map.setBaseLayer(tms_tk50_8);
    break;
  case 4:
    map.setBaseLayer(tms_tk50_4);
    break;
  });
});
```

Anhang F: Proxyskript

s. Abschnitte 4.2 WFS, 4.3 GeoRSS, 7.3 Passwort im Proxyskript

```
#!/usr/bin/env python

"""This is a blind proxy that we use to get around browser
restrictions that prevent the Javascript from loading pages not on the
same server as the Javascript. This has several problems: it's less
efficient, it might break some sites, and it's a security risk because
people can use this proxy to browse the web and possibly do bad stuff
with it. It only loads pages via http and https, but it can load any
content type. It supports GET and POST requests."""

import urllib2
import cgi
import sys, os

# Designed to prevent Open Proxy type stuff.

allowedHosts = ['www.openlayers.org', 'openlayers.org',
                 'geoentw.bvv.bayern.de:8080',
                 'labs.metacarta.com', 'world.freemap.in',
                 'geodaten.bvv.bybn.de',
                 'prototype.openmnd.org', 'geo.openplans.org',
                 'sigma.openplans.org', 'api.flickr.com',
                 'www.openstreetmap.org']

method = os.environ["REQUEST_METHOD"]

if method == "POST":
    qs = os.environ["QUERY_STRING"]
    d = cgi.parse_qs(qs)
    if d.has_key("url"):
        url = d["url"][0]
    else:
        url = "http://www.openlayers.org"
else:
    fs = cgi.FieldStorage()
    url = fs.getvalue('url', "http://www.openlayers.org")

try:
    host = url.split("/")[2]
    if allowedHosts and not host in allowedHosts:
        print "Status: 502 Bad Gateway"
        print "Content-Type: text/plain"
        print
        print "This proxy does not allow you to access that location."
        print
        print os.environ

    elif url.startswith("http://") or url.startswith("https://"):

        if method == "POST":
            length = int(os.environ["CONTENT_LENGTH"])
            headers = {"Content-Type": os.environ["CONTENT_TYPE"]}
            body = sys.stdin.read(length)
            r = urllib2.Request(url, body, headers)
            y = urllib2.urlopen(r)
```

```

else:
    headers = {"Authorization": "Basic c2NobWl0dDp1cmxhdWI="}
    r = urllib2.Request(url, None, headers)
    y = urllib2.urlopen(r)

    # print content type header
    i = y.info()
    if i.has_key("Content-Type"):
        print "Content-Type: %s" % (i["Content-Type"])
    else:
        print "Content-Type: text/plain"
    print

    print y.read()

    y.close()
else:
    print "Content-Type: text/plain"
    print
    print "Illegal request."

except Exception, E:
    print "Status: 500 Unexpected Error"
    print "Content-Type: text/plain"
    print
    print "Some unexpected error occurred. Error text was:", E

```

Anhang G: WFS-Punktobjekte

s. Abschnitt 4.2.1

```
var ls = new OpenLayers.Control.LayerSwitcher();
var options = {
    resolutions: [150,100,40,12.5,10,4,2.1166667],
    maxExtent: new OpenLayers.Bounds(4281000, 5235000, 4640000, 5606000),
    units: 'm',
    projection: "EPSG:31468",
    tileSize: new OpenLayers.Size(500,500),
    controls:[
        new OpenLayers.Control.PanZoomBar,
        ls,
        new OpenLayers.Control.Permalink('permalink'),
        new OpenLayers.Control.MousePosition,
        new OpenLayers.Control.ScaleLine,
        new OpenLayers.Control.MouseToolbar
    ]
};

    var map = new OpenLayers.Map( "map", options );

// Proxy für WFS
OpenLayers.ProxyHost = "/cgi-bin/proxy.cgi?url=";

// Anlegen der Ebenen

var tk = new OpenLayers.Layer.WMS("TK",
    "http://geodaten.bvv.bybn.de/ogc/ogc_dipl.cgi?",
    {layers: 'tk', format: 'image/png', srs:'EPSG:31468', version:'1.1.1'},
    {displayInLayerSwitcher: false});

// Erstellen der Styles für den Punktmarker
var style1 = OpenLayers.Util.extend({},
    OpenLayers.Feature.Vector.style['default']);
style1.strokeWidth = 2;
style1.strokeColor = "#ff0000";
style1.fillOpacity = 1;

var style1_selected = OpenLayers.Util.extend({},
    OpenLayers.Feature.Vector.style['default']);
style1_selected.strokeWidth = 2;
style1_selected.strokeColor = "#ffff00";
style1_selected.fillOpacity = 1;

var wfs1 = new OpenLayers.Layer.WFS("WFS Landkreise",
    "http://geoentw.bvv.bayern.de:8080/geoserver/wfs",
    {typename: 'lkreise'},
    {extractAttributes: true, style: style1});

var options = {hover: true,
    onSelect: function(feature){
        popup = new OpenLayers.Popup.FramedCloud(
            "chicken", // ID
            feature.geometry.getBounds().getCenterLonLat(), // lonlat
            null, // autosized
            "Landkreis: <b>" + feature.attributes.lkr + "</b>"
        );
    }
};
```



```

        <br>Regierungsbezirk: <b>" + feature.attributes.rbez+"</b>",
        // contentHTML
        null, // anchor
        false), // closeBox
        map.addPopup(popup)},
        selectStyle: style1_selected,

// Geht ohne Feature nicht!! "The function should expect to be called with a
feature." Source: SelectFeature.js
        onUnselect: function(feature){selectStyle: style1,
            popup.destroy()}
        }

        var select = new OpenLayers.Control.SelectFeature(wfs1, options);
        map.addControl(select);

// Erstellen der Marker

        var symbolizer_gold = OpenLayers.Util.applyDefaults({
            externalGraphic: "OpenLayers-2.7/img/marker-gold.png",
            pointRadius: 12, fillOpacity: 1},
            OpenLayers.Feature.Vector.style["default"]);
        var symbolizer_blue = OpenLayers.Util.applyDefaults({
            externalGraphic: "OpenLayers-2.7/img/marker-blue.png",
            pointRadius: 12, fillOpacity: 1},
            OpenLayers.Feature.Vector.style["default"]);
        var symbolizer_green = OpenLayers.Util.applyDefaults({
            externalGraphic: "OpenLayers-2.7/img/marker-green.png",
            pointRadius: 12, fillOpacity: 1},
            OpenLayers.Feature.Vector.style["default"]);
        var point_symbolizer = OpenLayers.Util.applyDefaults({pointRadius: 6,
            fillColor: '000000', fillOpacity: 1,
            strokeWidth: 7, strokeColor: '#329cff'},
            OpenLayers.Feature.Vector.style["default"]);

// Anlegen der WFS-Ebenen

        var wfs2 = new OpenLayers.Layer.WFS("WFS Regierungsbezirke",
            "http://geontw.bvv.bayern.de:8080/geoserver/wfs",
            {typename: 'rbezirke_wfs'},
            {styleMap: new OpenLayers.StyleMap({"default": symbolizer_gold})});

        var wfs3 = new OpenLayers.Layer.WFS("WFS Gemeinden",
            "http://geontw.bvv.bayern.de:8080/geoserver/wfs",
            {typename: 'gemeinden'},
            {styleMap: new OpenLayers.StyleMap({"default": symbolizer_green}),
            minResolution: 2.1166667, maxResolution: 10});

        var wfs4 = new OpenLayers.Layer.WFS("WFS Gemeindeteile",
            "http://geontw.bvv.bayern.de:8080/geoserver/wfs",
            {typename: 'gde_teil'},
            {styleMap: new OpenLayers.StyleMap({"default": point_symbolizer}),
            minResolution: 2.1166667, maxResolution: 10});

// Hinzufuegen der Layer zu der Karte

        map.addLayers([tk, wfs1, wfs2, wfs3, wfs4]);
        select.activate();

        ls.maximizeControl();
        map.setCenter(new OpenLayers.LonLat(4402829, 5510801));

```

Anhang H: WFS-Polygone

s. Abschnitt 4.2.2

```
var ls = new OpenLayers.Control.LayerSwitcher({activeColor: 'white', fontColor:
'black'});
var options = {
  resolutions: [12.5, 10, 4, 2.1166667, 1, 0.5, 0.25, 0.1],
  maxExtent: new OpenLayers.Bounds(4281000, 5235000, 4640000, 5606000),
  units: 'm',
  projection: "EPSG:31468",
  tileSize: new OpenLayers.Size(500,500),
  controls:[
    new OpenLayers.Control.PanZoomBar,
    ls,
    new OpenLayers.Control.Permalink('permalink'),
    new OpenLayers.Control.MousePosition,
    new OpenLayers.Control.Navigation(),
    new OpenLayers.Control.ScaleLine      ]
};

var map = new OpenLayers.Map( "map", options );

// Proxy für WFS
OpenLayers.ProxyHost = "/cgi-bin/proxy.cgi?url=";

// Anlegen der Ebenen

var tk = new OpenLayers.Layer.WMS("TK",
"http://geodaten.bvv.bybn.de/ogc/ogc_dipl.cgi?",
{layers: 'tk', format: 'image/png', srs:'EPSG:31468', version:'1.1.1'});

var dfk = new OpenLayers.Layer.WMS("DFK",
"http://igdb-as2.rz-nord.bayern.de:8080/cgi-bin/dfkwmselect?",
{layers: 'dfk', format: 'image/png', srs: 'EPSG:31468',
version: '1.1.1'});

var style1 = OpenLayers.Util.extend({},
OpenLayers.Feature.Vector.style['default']);
style1.strokeWidth = 2;
style1.strokeStyle = "#FF0000";
style1.fillColor = "#000000";
style1.fillOpacity = 1;

var style1_selected = OpenLayers.Util.extend({},
OpenLayers.Feature.Vector.style['default']);
style1_selected.strokeWidth = 2;
style1_selected.strokeStyle = "#ffff00";
style1_selected.fillColor = "#000000";
style1_selected.fillOpacity = 1;

var wfs = new OpenLayers.Layer.WFS("WFS Adressen",
"http://geontw.bvv.bayern.de:8080/geoserver/wfs",
{typename: 'app:BY-Hauskoordinaten'},
{extractAttributes: true, style: style1,
maxResolution: 0.1, minResolution: 0.1});
```

```

var options = {hover: true,
  onSelect: function(feature){
    popup = new OpenLayers.Popup.FramedCloud("chicken", // ID
    feature.geometry.getBounds().getCenterLonLat(), // lonlat
    null, // autosized
    feature.attributes.strassenname + " " +
    feature.attributes.hausnummer + "<br>" +
    feature.attributes.postalischerName, // contentHTML
    null, // anchor
    false), // closeButton
    map.addPopup(popup)},
  selectStyle: style1_selected,
  onUnselect: function(feature){selectStyle: style1,
  popup.destroy()}
}

var select = new OpenLayers.Control.SelectFeature(wfs, options);
map.addControl(select);

var wfs_poly = new OpenLayers.Layer.WFS("WFS Polygone",
  "http://geontw.bvv.bayern.de:8080/geoserver/wfs",
  {typename: 'v_shape_axflst'},
  {maxResolution: 0.25, minResolution: 0.1});
var select_poly = new OpenLayers.Control.SelectFeature(wfs_poly);
map.addControl(select_poly);

var select_toggle = new OpenLayers.Control.ToggleSelect({
  displayClass: "mbControlSelect", title: "Selektion"});
var panel = new OpenLayers.Control.Panel({displayClas: "olControlPanel"});
panel.addControls([select_toggle]);
map.addControl(panel);

// Hinzufügen der Layer zu der Karte

map.addLayers([dfk, tk, wfs, wfs_poly]);
select.activate();

ls.maximizeControl();

map.setCenter(new OpenLayers.LonLat(4295658, 5530801), 3);

```

Anhang I: GeoRSS

s. Abschnitt 4.3

```
var map = new OpenLayers.Map('map', {maxResolution:'auto'});
var layer = new OpenLayers.Layer.WMS( "OpenLayers WMS",
    "http://labs.metacarta.com/wms/vmap0", {layers: 'basic'} );
map.addLayer(layer);

map.setCenter(new OpenLayers.LonLat(0, 0), 0);
map.addControl(new OpenLayers.Control.LayerSwitcher());

OpenLayers.ProxyHost = "/cgi-bin/proxy.cgi?url=";
// da Ajax Zugriff auf andere Domäne verhindert
// Erstellen des GML-Layers mit GeoRSS-Format und styleMap

var markerLayer = new OpenLayers.Layer.GML("Fotos von Flickr",
    "http://api.flickr.com/services/feeds/geo/?lang=de-de&format=rss2",
    {format: OpenLayers.Format.GeoRSS, formatOptions: {
        // Miniaturbild als Attribut des Features
        createFeatureFromItem: function(item) {
            var feature =
OpenLayers.Format.GeoRSS.prototype.createFeatureFromItem.apply(this, arguments);
            feature.attributes.thumbnail =
this.getElementsByTagNameNS(item, "*", "thumbnail")[0].getAttribute("url");
            return feature;
        }
    },
    // Beim Anklicken des Bildes, wird dieses vergrößert
    styleMap: new OpenLayers.StyleMap({
        "default": new OpenLayers.Style({externalGraphic: "${thumbnail}",
            pointRadius: 25}),
        "select": new OpenLayers.Style({pointRadius: 40})
    })
});
map.addLayer(markerLayer);

var georss = new OpenLayers.Layer.GeoRSS("Name",
    "http://api.flickr.com/services/feeds/geo/?lang=de-de&format=rss2");
map.addLayer(georss);

// Beim Anklicken des Bildes wird außerdem ein Popup gezeigt
var popup;
var popupControl = new OpenLayers.Control.SelectFeature(markerLayer, {
    onSelect: function(feature) {var pos = feature.geometry;
        if (popup) {
            map.removePopup(popup);
        }
        popup = new OpenLayers.Popup.FramedCloud("popup",
            new OpenLayers.LonLat(pos.x, pos.y),
            null,
            "<b>" + feature.attributes.title + "</b>" +
            feature.attributes.description,
            null,
            false);
        map.addPopup(popup); },
    onUnselect: function(feature) {popup.destroy();} });
map.addControl(popupControl);
popupControl.activate();
```

Anhang J: Erfassen und Editieren

s. Kapitel 6

```
var tk = new OpenLayers.Layer.WMS("TK" ,
    "http://geodaten.bvv.bybn.de/ogc/ogc_dipl.cgi?",
    {layers: 'tk', format: 'image/png', srs: 'EPSG:31468', version: '1.1.1'});
var vlayer = new OpenLayers.Layer.Vector("Editieren");
var drag = new OpenLayers.Control.DragFeature(vlayer);
var toolb = new OpenLayers.Control.EditingToolbar(vlayer);
toolb.addControls(drag);

var ls = new OpenLayers.Control.LayerSwitcher();
var options = {
    resolutions: [80, 60, 40, 30, 20, 12.5, 10, 4, 2.1166667],
    maxExtent: new OpenLayers.Bounds(4281000, 5235000, 4640000, 5606000),
    units: 'm',
    projection: "EPSG:31468",
    tileSize: new OpenLayers.Size(500,500),
    controls: [
        new OpenLayers.Control.PanZoomBar,
        ls,
        new OpenLayers.Control.Permalink('permalink'),
        new OpenLayers.Control.MousePosition,
        new OpenLayers.Control.Navigation(),
        new OpenLayers.Control.ScaleLine,
        toolb]
};

var map = new OpenLayers.Map( "map", options );

// Hinzufuegen der Layer zu der Karte

map.addLayers([tk, vlayer]);

ls.maximizeControl(); // LayerSwitcher beim Laden der Seite maximiert

map.setCenter(new OpenLayers.LonLat(4295658, 5530801));

var wkt = new OpenLayers.Format.WKT();

var options = {hover: true,
    onSelect: function(feature){popup = new
        OpenLayers.Popup.FramedCloud("chicken", // ID
        feature.geometry.getBounds().getCenterLonLat(),
        // lonlat
        null, // autosized
        "<b>WKT:</b> " + wkt.write(feature) + "<br>
        <b>Länge:</b> " + feature.geometry.getLength(),
        // contentHTML
        null, // anchor
        false), // closeBox
        map.addPopup(popup)},
    onUnselect: function(feature){popup.destroy()}
}

var select = new OpenLayers.Control.SelectFeature(vlayer, options);
map.addControl(select);
select.activate();
```

Anhang K: Passwortschutz

s. Kapitel 7

```
var options = {
    resolutions: [150,100,40,12.5,10,4,2.1166667],
    maxExtent: new OpenLayers.Bounds(4281000, 5235000, 4640000, 5606000),
    units: 'm',
    projection: "EPSG:31468",
    tileSize: new OpenLayers.Size(500,500),
    controls: [
        new OpenLayers.Control.PanZoomBar(),
        new OpenLayers.Control.LayerSwitcher(),
        new OpenLayers.Control.Permalink('permalink'),
        new OpenLayers.Control.MousePosition(),
        new OpenLayers.Control.ScaleLine(),
        new OpenLayers.Control.MouseToolbar()
    ]
};

map = new OpenLayers.Map( "map", options );
wms = new OpenLayers.Layer.WMS( "WMS",
    "http://schmitt:urlaub@geodaten.bvv.bybn.de/ogc/ogc_dipl.cgi?",
    {layers: 'tk', format: 'image/png', srs:'EPSG:31468',
    version:'1.1.1' });

// passwortgeschützter WMS
// headers: {"Authorization": "Basic c2NobWl0dDp1cmxhdWI="}});
// "/cgi-bin/proxy.cgi?url=http://geodaten.bvv.bybn.de/ogc/ogc_dipl_auth.cgi?"

wms_auth = new OpenLayers.Layer.WMS( "WMS Auth", "/cgi-bin/proxy.cgi?url=http%3A%2F%2Fgeodaten%2Ebv%2Ebybn%2Ede%2Fogc%2Fogc%5Fdipl%5Fauth%2Ecgi%3F",
    // http://geodaten.bvv.bybn.de/ogc/ogc_dipl_auth.cgi?
    {layers: 'tk', format: 'image/png', srs:'EPSG:31468', version:'1.1.1'},
    {authentication: true});

// Hinzufügen der Layer zu der Karte
map.addLayers([wms, wms_auth]);

var RW = 4468442;
var HW = 5336771;
map.setCenter(new OpenLayers.LonLat(RW, HW));
```


Glossar

Ajax

Ajax steht für *Asynchronous Javascript and XML* und ist eine Schlüsseltechnik im →Web 2.0. Ajax gibt es seit ca. Februar 2005. Kernstück ist der *XMLHttpRequest*, der das dynamische Nachladen von Internetseiten ermöglicht. Damit muss nicht mehr die komplette Seite aktualisiert werden. Mit Hilfe von Ajax werden Desktopanwendungen im Internet nachgebildet.

Amazon S3

Amazon S3 steht für *Amazon Simple Storage Service* und ist einer der *Amazon Web Services* (AWS). Er dient der Datenspeicherung im Internet. Weitere Informationen findet man unter <http://aws.amazon.com/s3> [25.11.08].

API

API steht für *Application Programming Interface*, was auf Deutsch *Schnittstelle für Anwendungsprogrammierung* heißt. Über diesen Zugang ermöglicht man Programmierern die Einbindung des Softwaresystems in die eigene Anwendung.

BSD-Lizenz

Software unter der BSD-Lizenz darf frei verwendet werden. Sie darf sogar ohne Quelltextveröffentlichung verändert und verbreitet werden. Einzige Bedingung ist, den originalen Copyright-Vermerk nicht zu entfernen.

CC-by-sa

Creative Commons (CC, engl. *schöpferisches Gemeingut*) ist eine gemeinnützige Gesellschaft, die Standard-Lizenzverträge

veröffentlicht. Bei einer *CC-by-sa*-Lizenz muss der Name des Autors genannt werden (*by*) und das Werk muss, auch nach Veränderungen, unter der gleichen Lizenz weitergegeben werden (*sa*).

CGI

Das *Common Gateway Interface* (engl. *Allgemeine Zugangs-Schnittstelle*) ist ein →W3C-Standard für den Datenaustausch zwischen einem Webserver und dritter Software, die Anfragen bearbeitet. CGI verwendet man, um Webseiten oder Daten dynamisch zu generieren.

CSS

CSS bezeichnet *Cascading Stylesheets*. Die *kaskadierten Formatvorlagen* sind eine Ergänzung zur Auszeichnungssprache HTML. Sie beschreiben die Formatierung der Internetseite, wogegen HTML die Struktur beschreibt.

DOM

Das *Document Object Model* ist eine Spezifikation des →W3C. Sie definiert den Zugriff auf HTML- oder →XML-Dokumente und deren Objekte.

EPSG

EPSG steht für *European Petroleum Surveying Group*. Die Arbeitsgruppe wurde 2005 durch das *Surveying and Positioning Committee der International Association of Oil and Gas Producers* (OGP) abgelöst. Sie katalogisiert weltweit Koordinatensysteme mit eindeutigen Schlüsselnummern, den EPSG-Codes. Weitere Informationen befinden sich auf der Homepage www.epsg.org [25.11.08].

Framework

Ein Framework stellt ein Programmiergerüst dar, das mächtige Funktionen oder Klassen für den Aufbau einer Anwendung liefert.

GeoServer

GeoServer ist eine Open-Source-Software, die die Dienste WMS, WFS und WCS bereitstellt. Die Serversoftware ist in Java entwickelt. Weitere Informationen findet man auf der offiziellen *GeoServer*-Webseite geoserver.org.

Geography Markup Language (GML)

GML dient der Modellierung, dem Transport und der Speicherung von raumbezogenen Objekten. Die Sprache basiert auf XML. GML ist eine offizielle internationale Norm. Sie wird vom OGC gemeinsam mit dem technischen Komitee der ISO festgelegt.

HTTP

HTTP steht für *Hypertext Transfer Protocol*, was auf Deutsch *Hypertext-Übertragungsprotokoll* heißt. Das Protokoll dient der Datenübertragung in einem Netzwerk und wird hauptsächlich für Webseiten im Internet verwendet. Im OSI-Schichtmodell ist es den anwendungsorientierten Schichten 5 bis 7 zugeordnet.

Java-Applet

Ein *Java-Applet* ist ein Programm, das in *Java* geschrieben ist und im Webbrowser ausgeführt wird. Dazu benötigt der Browser eine Laufzeitumgebung, die entweder Teil des Browsers ist meistens aber nachträglich als Plugin installiert werden muss.

JavaScript

JavaScript ist eine Skriptsprache und dient der Ergänzung von HTML. Ihr Sprachkern ist als *ECMAScript 262* (www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf [11.12.08]) standardisiert. Sie wird zur Laufzeit vom Webbrowser interpretiert. Dazu benötigen die Browser entsprechende Interpreter-Software. Allerdings sind die *JavaScript*-Funktionen bei den derzeitigen verfügbaren Browsern oft nicht vollständig in der Interpreter-Software umgesetzt. Außerdem muss beachtet werden, dass der Benutzer das Ausführen von *JavaScript* deaktivieren kann. Aus Sicherheitsgründen wird *JavaScript* in einer *Sandbox* ausgeführt, um einen direkten Zugriff auf den Rechner zu verhindern.

Leetspeak

Leetspeak ist vom englischen *elite* abgeleitet und bezeichnet das Ersetzen von Buchstaben durch ähnlich aussehende Ziffern sowie Sonderzeichen. So wird Leetspeak selbst *1337* geschrieben.

Landesamt für Vermessung und Geoinformation

Dem *Bayerischen Landesamt für Vermessung und Geoinformation* mit Hauptsitz in München ist die Abteilung *Bayerische Vermessungsverwaltung (BVV)*, *Informations- und Kommunikationstechnik* des Bayerischen Staatsministeriums der Finanzen übergeordnet. Dem LVG wiederum gehören die bayerischen Vermessungsämter an (vgl. BVV 2008a). Diese führen das Liegenschaftskataster, das mit dem Grundbuch die Eigentumsverhältnisse sichert (vgl. BVV 2008b).

Als Dienstleistungsunternehmen stellt das LVG raumbezogene Basisdaten, den SAPOS®-Dienst und den *BayernViewer* für das gesamte Staatsgebiet Bayerns von 70 000 km² der Öffentlichkeit zur Verfü-

gung. Die Basisdaten umfassen Orthophotos, digitale Geländemodelle, topographische Karten und Geoinformationssysteme, wie zum Beispiel ATKIS®. 600 Mitarbeiter sind im LVG beschäftigt, wovon 108 der Abteilung 4 *Informations- und Kommunikationstechnik Entwicklung* zugeteilt sind. Das Referat 42 *Online-Dienste, Basiskomponente GeodatenOnline* dieser Abteilung umfasst 15 Mitarbeiter. Dipl.-Ing. (FH) Manfred Zerndl arbeitet im Referat 42 und war vor Ort der Betreuer der Diplomarbeit.

OGC

OGC ist die Abkürzung für *Open Geospatial Consortium, Inc.* Der 1994 gegründeten Organisation gehören über 200 internationale Mitglieder aus den Bereichen Industrie, Verwaltung und Forschung an. Um die Interoperabilität zwischen Geoinformationssystemen, sowie zwischen diesen Systemen und Standardsoftware zu erreichen, definiert sie Standards wie WMS, WFS, GML und WKT.

Open Source Geospatial Foundation

Die *Open Source Geospatial Foundation* ist eine gemeinnützige Organisation zur Förderung der Entwicklung und Nutzung von freier und Open-Source-Software in der räumlichen Datenverarbeitung. Die Homepage der Stiftung findet man unter www.osgeo.org [25.11.08].

PostGIS

PostGIS erweitert die objekt-relationale Datenbank →*PostgreSQL* um geografische Objekte und Funktionen. Es ist ein freies geografisches Informationssystem.

PostgreSQL

PostgreSQL ist ein objektrelationales Datenbankmanagementsystem. Es ist weitgehend konform mit dem SQL-Standard

ANSI-SQL 92. *PostgreSQL* ist unter der →BSD-Lizenz frei verfügbar. Die Homepage befindet sich unter postgresql.org.

Prototype.js

Prototype ist ein freies →*JavaScript*-Framework, das verschiedene Programmierhilfen zur Verfügung stellt. Es beinhaltet Funktionen für Klassenunterstützung und *XML-HttpRequests*. Weitere Informationen findet man unter www.prototypejs.org [25.11.08].

Rico

Rico ist ein open source →*JavaScript*-Framework auf Grundlage von →*Prototype*. Sie beinhaltet Funktionen wie Animationseffekte, Tabellen und Drag und Drop. openrico.org ist die Homepage von *Rico*.

RSS

RSS steht in der Version 2.0 für *Really Simple Syndikation*. Der RSS ist ein Webdienst, ähnlich einem Newsticker. Man kann einen sogenannten *RSS-Feed* abonnieren und wird dann automatisch über neue Einträge informiert.

RSS ist ein *GML-Profil*, d. h. dass GML durch ein Schema logisch eingeschränkt wurde.

SVG

Scaleable Vector Graphics (Skalierbare Vektorgrafiken) ist ein Vektorgrafikstandard des →W3C. SVG ist ein →XML-basiertes Dateiformat und ergänzt damit den HTML-Standard um die Darstellung komplexer Vektordaten. Mit ihm ist es auch möglich, geometrische Figuren transparent zu schalten. SVG wird von allen Browsern unterstützt außer vom *Internet Explorer*. Dieser nutzt stattdessen sein eigenes Format →VML.

UMN MapServer

Die Open-Source-Software dient der Darstellung von raumbezogenen Geodaten im Internet. Sie unterstützt die →OGC Standards WMS, WFS und WCS. Das Programm wird vom Webserver über →CGI aufgerufen.

VML

Die *Vector Markup Language* beschreibt wie →SVG zweidimensionale Vektorgrafiken in →XML. VML trug zwar zum →W3C-Standard →SVG bei, wurde aber als eigener Standard abgelehnt. Dennoch implementiert Microsoft weiterhin VML und nicht →SVG im *Internet Explorer*.

Web 2.0

Früher (*Web 1.0*) bestand das Internet aus statischen Seiten und es gab eine klare Trennung zwischen Benutzer und Anbieter. Heute (*Web 2.0*) gibt es viele interaktive und kollaborative Elemente im Internet. Web-Anwendungen werden wie Desktop-Anwendungen gestaltet. Benutzer können Inhalte selbst bearbeiten, zum Beispiel in

→*Wikis*. Aus technischer Sicht bezeichnet *Web 2.0* eine Anzahl von Methoden wie *Web-Service-APIs*, →*Ajax*, und Abonnement-Dienste wie →*RSS*.

Wiki

Ziel eines Wiki (hawaiisch für *schnell*) ist das Wissen von den Autoren kollaborativ auszudrücken. Die Inhalte können von den Benutzern gelesen und geändert werden.

W3C

Das *World Wide Web Consortium* ist eine Organisation, die die Weiterentwicklung technischer Standards im Internet koordiniert.

XML

XML steht für *Extensible Markup Language*, zu Deutsch *erweiterbare Auszeichnungssprache*. Der Standard vom →W3C dient der Beschreibung, Validierung und dem Austausch von Daten. Wobei die Beschreibung hierarchisch strukturiert ist und in Form von Textdaten mit der Dateierweiterung *.xml* gespeichert ist.

Literaturverzeichnis

BVV 2008a: Bayerische Vermessungsverwaltung, Wir über uns - Organisation, www.geodaten.bayern.de, Datum des Zugriffs: 10. Oktober 2008

BVV 2008b: Bayerische Vermessungsverwaltung (2008): Flyer "Auftrag-Geschichte-Produkte", www.geodaten.bayern.de - Wir über uns, Datum des Zugriffs: 10. Oktober 2008

Eider 2008: Eider Carsten, GeoEditor, <https://143.93.78.98/geoEditor2/editor.html>, Datum des Zugriffs: 15. Dezember 2008

OGC 2002: Open Geospatial Consortium Inc., Web Map Service Implementation Specification - Version 1.1.1, http://portal.opengeospatial.org/files/?artifact_id=1081&version=1&format=pdf, Datum des Zugriffs: 10. Dezember 2008

OGC 2007: Open Geospatial Consortium Inc., OpenGIS Tiled WMS Discussion Paper, <http://www.opengeospatial.org/standards/dpd>

OL 2008: OpenLayers, OpenLayers: Home, www.openlayers.org, Datum des Zugriffs: 14. Oktober 2008

OL Bsp 2008a: OpenLayers, OpenLayers Spherical Mercator Example, <http://www.openlayers.org/dev/examples/spherical-mercator.html>, Datum des Zugriffs: 27. Oktober 2008

OL Bsp 2008b: OpenLayers, Measure Example, <http://www.openlayers.org/dev/examples/measure.html>, Datum des Zugriffs: 21. November 2008

OL Bsp 2008c: OpenLayers, GeoRSS from Flickr in OpenLayers, <http://www.openlayers.org/dev/examples/georss-flickr.html>, Datum des Zugriffs: 26. November 2008

OL FAQ 2008: Open Layers, Why don't vectors work in \$browser?, <http://faq.openlayers.org/vector-related-questions/why-dont-vectors-work-in-browser>, Datum des Zugriffs: 19. November 2008

OL MailingList 2008a: Schmidt Christopher, OpenLayers - Nabble - OpenLayers Users - OpenLayers Question: Google Physical Projections and epsg:26915, <http://www.nabble.com/OpenLayers-Question%3A-Google-Physical-Projections-and-epsg%3A26915-td18148318.html#a18155050>, Datum des Zugriffs: 22. Oktober 2008

OL MailingList 2008b: Schmidt Christopher, OpenLayers - Nabble - MousDefaults with yahoo map, <http://www.nabble.com/MousDefaults-with-yahoo-map-tt19252009.html#a19450500>, Datum des Zugriffs: 23. Oktober 2008

OL MailingList 2008c: Martijn van Oosterhout, Nabble - TileCache - whether to use tilecache to serve tiles, <http://www.nabble.com/whether-to-use-tilecache-to-serve-tiles-td15088292.html#a15132675>, Datum des Zugriffs: 13. November 2008

OL Wiki 2008a: OpenLayers, Map24 Layer in OpenLayers, <http://trac.openlayers.org/wiki/Layer/Map24>, Datum des Zugriffs: 27. Oktober 2008

OL Wiki 2008b: OpenLayers, Configuring ZoomLevels in OpenLayers, <http://trac.openlayers.org/wiki/SettingZoomLevels>, Datum des Zugriffs: 29. Oktober 2008

OL Wiki 2008c: OpenLayers, Ways to Optimization OpenLayers, <http://trac.openlayers.org/wiki/OpenLayersOptimization>, Datum des Zugriffs: 4. November 2008

OL Wiki 2008d: OpenLayers, Using Custom Tile Sources / Google-like Tile Layer Support, <http://trac.openlayers.org/wiki/UsingCustomTiles>, Datum des Zugriffs: 13. November 2008

OSGeo 2007: Open Source Geospatial Foundation, TilingStandard, <http://wiki.osgeo.org/wiki/TilingStandard>, Datum des Zugriffs: 5. November 2008

Python 2008: Python Library Reference, 18.6 urllib2 - extensible library for opening URLs, <http://docs.python.org/lib/module-urllib2.html>, Datum des Zugriffs: 28. November 2008

Ramm & Topf 2008: Ramm F, J. Topf, *OpenStreetMap - Die freie Weltkarte nutzen und mitgestalten*, Lehmanns Media, Berlin, 1. Auflage

Schmidt C. 2007: Schmidt Christopher, Technical Ramblings - Blog Archive - Google Projection: 900913, <http://crschmidt.net/blog/243/google-projection-900913>, Datum des Zugriffs: 22. Oktober 2008

Schmidt C. 2008: Schmidt Christopher, Spherical Mercator, http://crschmidt.net/~crschmidt/spherical_mercator.html, Datum des Zugriffs: 23. Oktober 2008

TileCache 2008: MetaCarta, Installing TileCache, <http://www.tilecache.org/readme.html#description>, Datum des Zugriffs: 5. November 2008

Zorn 2006: Zorn Walter, JavaScript, DHTML: Linie, Ellipse, Kreis, Polygon, Dreieck, Rechteck, Polyline zeichnen, <http://www.walterzorn.de/jsgraphics/jsgraphics.htm>, Datum des Zugriffs: 18. November 2008

Erklärung

Die vorliegende Diplomarbeit wurde von mir zur Diplomprüfung im Wintersemester 2008 selbst verfasst und ohne fremde Hilfe angefertigt. Die verwendeten Hilfsmittel und Literaturquellen sind in den entsprechenden Verzeichnissen aufgeführt.

München, den 19. Dezember 2008

Carolin Wengerter

Gesehen:

Erstprüfer und Betreuer:

Datum

Prof. Dr.-Ing. Franz-Josef Behr

Zweitprüfer:

Datum

Prof. Dr.-Ing. Hardy Lehmkuhler