Hochschule Neubrandenburg
University of Applied Sciences

Masterarbeit

Aufsetzen eines Facility Management Systems für den Rostocker Fracht- und Fischereihafen Rostock unter Verwendung internetbasierter GIS-Technologien[1].

Eingereicht von *Balschmiter, Tim* am *14.Oktober 2009*

an der

Hochschule Neubrandenburg

1. Gutachter: Dr. Korduan, Peter
Universität Rostock, Professur Geodäsie und Geoinformatik,
Dozent an der Hochschule Neubrandenburg

2.Gutachter: Prof. Dr. Bill, Ralf
Universität Rostock, Professur Geodäsie und Geoinformatik,
Dozent an der Hochschule Neubrandenburg

[1] urn:nbn:de:gbv:519-thesis2009-0235-7

Danksagung

Ich bedanke mich bei den Mitarbeitern der Verwaltungsabteilung des Rostocker Fracht- und Fischereihafens, die es mir ermöglicht haben mein Masterarbeitsthema in ihrem Unternehmen umzusetzen.

Weiterer Dank gebührt Herrn Prof. Dr. Ralf Bill und Herrn Dr. Peter Korduan, die mich bereits in meiner Zeit an der Universität Rostock bei der Anfertigung meiner Bachelorarbeit unterstützt haben und mir auch während der Umsetzung meiner Masterarbeit für die Hochschule Neubrandenburg unterstützend zur Seite standen.

Abschließender Dank geht an meine Familie die, neben der Studienfinanzierung, immer für mich da waren.

Abstract

This Master's Thesis describes the development of a facility management system for the Rostocker Fracht- und Fischereihafen GmbH (RFH). The implemented facility management system simplifies processes and organizes redundant free data storage for the administration of the RFH. Therefore were merged two existing databases for property information and for geometric data. The realization includes web based GIS technologies and open source solutions. The resulting administrative database is accessible through a designed intranet web interface for all employees.

Inhaltsverzeichnis

Einleitung

Grundlagen

Software

Umsetzung

Fazit

Verzeichnisse

Anhang

Einleitung

1 Rostocker Fracht- und Fischereihafen GmbH

Die Rostocker Fracht- und Fischereihafen GmbH, im Weiteren als RFH bezeichnet, ist ein „zuverlässiger Dienstleister im Umschlag von Stückgut, Massengut und Projektladungen“[8] in der Hansestadt Rostock. 20 Liegeplätze stehen für kleinere und mittlere Handelsschiffe, aus dem vorwiegend baltischen Raum, an einem 2100 Meter langen Kai zur Verfügung. Den Handelspartnern werden, außer den Liegeplätzen, ein 8.000 m² großes Kühlhaus, direkte Verkehrsanbindung und Serviceleistungen angeboten.

Neben der Hafenwirtschaft ist die Verwaltung und Vermarktung der 52 ha großen Betriebsfläche ein wichtiges Aufgabenfeld des RFH. Im Juni 2009 nutzten rund 170 Firmen beziehungsweise Betriebe die Flurstücke und Immobilien des Rostocker Hafens. Für diesen Bereich hat die RFH GmbH eine eigene Verwaltungsabteilung, welche für die Erfassung und Wartung von Sach[2]- und Geometriedaten[3] zuständig sind.

1.1 Arbeitsablauf in der Verwaltung

Die Verwaltung von Flur-, Gebäude-, Raum- und sonstigen Daten wird im RFH durch die Mitarbeiterinnen Frau Paninka und Frau Clausen abgedeckt. Die beiden Damen sitzen in unterschiedlichen Büros, einer Etage im Hauptsitz der RFH GmbH. Neben den getrennten Arbeitsplätzen sind auch die Verwaltungsaufgaben beider Angestellten verschieden. Frau Paninkas Aufgabengebiet ist die Verwaltung und Pflege der Sachdaten. Dagegen arbeitet Frau Clausen die räumlichen Daten für die einzelnen Objekte in das vorhandene Geoinformationssystem ein. Dabei werden viele Sachdaten redundant aufgenommen und unabhängig von einander bearbeitet. Durch diesen Umstand entsteht eine Datenansammlung, bei der die Geodaten mit ihren Sachattributen nicht immer mit den Sachdaten des aktuellen Verwaltungsdatensatzes korrespondieren.

Möchte nun ein anderer Mitarbeiter des RFHs Auskünfte über ein Objekt einholen, muss dieser Frau Paninka kontaktieren, um die aktuellen Sachdaten zu erhalten. Soll den Sachdaten ein Lageplan oder eine einfache, visualisierte Objektkarte beigefügt werden, muss der Mitarbeiter zusätzlich das Büro von Frau Clausen aufsuchen. Im ungünstigsten Fall sind beide Damen nicht an ihrem Arbeitsplatz anzutreffen. Der Mitarbeiter, der mit den Daten eventuell einen potentiellen Kunden versorgen möchte oder Informationen für die Bilanz benötigt, muss nun darauf hoffen, dass sich beide Damen gleichzeitig in ihren Büros aufhalten. Aktuell ist die Auskunftsmöglichkeit nicht nur personen- und raumabhängig, sondern auch an spezielle

[2] Informationen, die ein Objekt näher beschreiben, z.B. Adresse oder Mieter eines Objektes
[3] Informationen, die ein Objekt in seiner Lage und Form beschreiben, siehe Kapitel 3.1

Software gebunden. Die Sachdaten sind in einer Microsoft Accessdatenbank auf dem Arbeitsplatzrechner von Frau Paninka hinterlegt. Frau Clausen speichert die geometrischen Daten im ESRI-Shapeformat auf ihrem Rechner ab. Um an alle Informationen zu gelangen, müssten, ohne das zuständige Personal, Bedienungskenntnisse über die auskunftsliefernden Softwarepakete vorhanden sein und der Zugang zu den informationsspeichernden Medien genehmigt werden.

1.2 Aufgabenstellung

Die Auftraggeber des RFHs wünschen sich ein Immobilien-Auskunftssystem. Dieses Auskunftssystem soll einigen Mitarbeitern den Zugang zu den Geodaten und den Sachdaten ermöglichen. Das Auskunft benötigende Personal, muss die Möglichkeit besitzen Karten eigenständig ausdrucken zu können und einfache Suchanfragen zu stellen. Mit Hilfe der Abfragen kann z.B. erfragt werden, welche Mietobjekte, in welchem Zeitraum, von wem zu welchem Preis gemietet sind, oder welche Mietobjekte, aktuell zu welchen Konditionen zur Verfügung stehen. Ebenso müssen Optionen zur Vermessung von Strecken und Flächen umgesetzt werden.

Im Vordergrund soll aber die ortsunabhängige und eigenständige Präsentation der Geometriedaten im Verbund mit den Sachdaten stehen. Wünschenswert von Seiten des RFHs ist eine verständliche und leicht zu verwaltende Benutzerumgebung.

1.3 Ziele und Ideen

Die Umsetzung des Facilitiy Management Systems nach Vorgaben der Verwaltungsmitarbeiter gewährt in der Entwicklung eine Menge Freiheiten. Die Gegebenheiten des RFHs sollen dabei maximal genutzt werden, um eine schnelle Integration des Auskunftssystems in den Arbeitsalltag zu ermöglichen. Die Arbeitsaufteilung, der in der Verwaltung zuständigen Damen, soll beibehalten werden. Auch die lokale Trennung darf bei der Nutzung der neuen Strukturen keine Rolle spielen. Weiterhin sollten sich Frau Paninka und Frau Clausen nicht in andere Softwarelösungen einarbeiten müssen, um einen reibungslosen Übergang vom bestehenden zum zukünftigen System zu schaffen. Das heißt Frau Paninka behält ihre Access-Datenbankoberfläche und Frau Clausen ihre ArcGIS-Oberfläche (als Hintergedanke bleibt festzuhalten, dass der Umstieg auf Open Source keine gravierenden Probleme bereiten würde). Die Änderungen, die im Verwaltungsablauf entstehen, spielen sich im Hintergrund ab. Die bearbeiteten Daten werden in Zukunft nicht mehr lokal auf dem Arbeitsplatzrechner gespeichert, sondern in einer zentralen Datenbank gesichert. Diese Datenbank wird als Schnittstelle zwischen den Geometriedaten und den

Sachdaten dienen. Frau Paninka verwaltet somit weiterhin die Sachdaten, während Frau Clausen die Geometrien der einzelnen RFH-Objekte betreut. Die doppelte Bearbeitung und Speicherung von Sachdaten entfällt durch die Bearbeitung einer gemeinsamen Datenmenge.
Das Auskunftssystem sollte, wenn möglich, jedem RFH-Mitarbeiter zur Verfügung stehen. Hier bietet sich der Einsatz von internetbasierter GIS-Technologie an. Anstelle des Internets wird das Intranet genutzt, welches identische Eigenschaften zum Internet aufweist. Es kommt zur Anwendung einer Server-Client-Struktur, das heißt der Server stellt die Daten für die Clients zur Verfügung. Der Abruf, der sich auf dem Server befindenden Daten, wird über den Browser stattfinden. Ein einfacher Browser dient also als Präsentationsplattform.
Das Zusammenführen von Verwaltung und Präsentation der RFH-Daten wird als Facility Management System bezeichnet.

Grundlagen

2 Facility Management System

Facility Mangement ist ein Begriff aus der Normung, der in der DIN EN 15221-1 definiert ist (siehe Definition 1) und vom Technischem Komitee CEN/TC348 entwickelt wurde.

Organisationen, wie es die Rostocker Fracht- und Fischerhafen GmbH ist, nutzen materielle Werte, Betriebsvermögen und Dienstleistungen um ihren Gewinn zu erwirtschaften. Die Norm des Facility Mangement unterstützt Unternehmen, deren Struktur aus den drei oben genannten Komponenten besteht. Der Zweck der DIN EN 15221 ist die Erschaffung einer eindeutigen und transparenten Kommunikation, sowie die effektive Nutzung von Ressourcen, wie es zum Beispiel die Arbeitskraft und das Dienstleistungsangebot sind. Auf dieser Basis soll das Angebot eine Qualitätssicherung und -steigerung erleben und Kommunikationsirritationen innerhalb der Organisation reduziert werden.

In der angesprochenen Norm sind für die Entwicklung eines Facility Management Systems für den RFH drei Definitionen von Bedeutung:

Facility Management:

„[Ist] die Integration von Prozessen innerhalb einer Organisation zur Erbringung und Entwicklung der vereinbarten Leistung, welche zur Unterstützung und Verbesserung der Effektivität der Hauptaktivitäten der Organisation dienen."

Bsp. im RFH:

Verbindung von Sach- und Geodaten über eine Datenbank, Vermeidung von Redundanz

Facility:

„[Ein] Facility ist ein materieller Vermögenswert, der eine Organisation unterstützt."

Bsp. im RFH:

Gebäude, Flurstücke und Räume

Facility Service:

„[Ist eine] Dienstleistung zur Unterstützung der Hauptaktivitäten einer Organisation, die von einem internen oder externen Leistungserbringer erbracht wird."

Bsp. im RFH:

Eingabe der Sachdaten in die Datenbank und Speicherung der Geodaten in der selbigen Datenbank.

Definition 1 Definitionen aus der DIN EN 15221-1 [1]

Die Erzeugung einer Präsentationsplattform und Schaffung der dafür nötigen Strukturveränderungen in der Verwaltung, sollen der Datenqualitätssicherung und -steigerung dienen.

3 Grundlegende Erläuterungen

3.1 Geodaten

Geodaten sind Daten, die an eine bestimmte Örtlichkeit gebunden sind. Solche raumbezogenen Informationen werden oft durch Koordinaten definiert, wie zum Beispiel der Eiffelturm befindet sich 48°51‘29‘‘[4] nördlicher Breite und 2°17‘35‘‘ östlicher Länge. Geodaten müssen nicht auf einen Punkt beschränkt sein, sondern können sich auch auf Linien, Flächen oder Körper beziehen. Eine Erweiterung der räumlichen Dimensionen ist durch einen Höhenwert oder einen Zeitwert möglich. So zum Beispiel in der Information: „In der Alpenregion lag zwischen Oktober und Februar, ab einer Höhe von 1500 Metern durchgängig Schnee.“

Die Informationen der Geodaten sind dabei nicht auf die Erde beschränkt, sondern können an jeden Punkt des Universums gebunden sein.

Im RFH sind die Geodaten, die durch Flächen beschriebenen Räume, Gebäude und Flurstücke die durch Sachattribute an Informationsgehalten gewinnen.

3.2 Geoinformationssysteme

Geoinformationssysteme[5] sind Verbunde, bestehend aus Hardware, Software und Daten, die es ermöglichen Geodaten zu erfassen, zu speichern, zu analysieren und darzustellen.

Es gibt verschiedene Arten von Geoinformationssysteme, wie z.B. Desktop-GIS, Internet-GIS und mobile GIS. Auf Grund der Verwendung im RFH werden das Desktop GIS und das internetbasierte GIS kurz erklärt.

3.2.1 Desktop GIS

Ein Desktop GIS besteht aus einem Arbeitsplatzrechner und einer lokal installierten Software. Es ist durch einen hohen Funktionsumfang gekennzeichnet.

3.2.2 Internetbasierte GIS

Anders als beim Desktop GIS sind internetbasierte GIS-Technologien nicht auf einem Arbeitsplatzrechner installiert, sondern ermöglichen über eine serverseitige Installation vielen Anwendern den Zugang. Neben einem Webserver wird auch ein Mapserver vorausgesetzt. Auf die nötigen server- und clientseitigen Softwarepakete wird ab Kapitel 5 eingegangen.

Neben den erhöhten Voraussetzungen für Internet-GIS sind die Funktionen im Normalfall deutlich eingeschränkter. Die meisten derartigen GIS sind auf die reine Datenausgabe

[4] Gesprochen als 48 Grad, 51 Minuten und 29 Sekunden
[5] GIS

spezialisiert, doch auch die Dateneingabe ist über das Internet möglich. Die Datenanalyse hält erst mit den Erneuerungen der Browsertechnologien ein, denn die dafür nötigen Programmfunktionen erfordern erhöhten, clientseitigen Rechenbedarf.

3.3 Koordinatensysteme

Koordinatensysteme beschreiben die Lage eines Punktes im Raum. Auf die unterschiedlichen Eigenschaften der verschiedenen Koordinatensysteme wird hier nicht näher eingegangen.
Im RFH wird das Koordinatensystem Pulkovo 1942 (83) der Gauss Krüger Zone 4 verwendet, welches auf dem Krasowsky Ellipsoid aufbaut. Im weiteren Verlauf wird lediglich der EPSG[6]-Code 2398 für das Koordinatensystem verwendet.
Weitere Abkürzungen die den EPSG-Code verlangen sind SRID[7] und SRS[8].

3.4 Open Geospatial Consortium -Standards

Die OGC-Standards sind Richtlinien, die das universelle Einbinden von Geodaten in beliebige Softwaresysteme ermöglicht. Zu den Standards zählen unter anderem der Web Map Service und der Web Feature Service, die auf Grund ihrer zukünftigen Verwendung im RFH im Kapitel 9 näher beschrieben werden.

[6] European Petroleum Survey Group
[7] Spatial Reference System Identifier
[8] Spatial Reference Systems

4 Programmiersprachen

Im Folgenden werden die für die Programmierung genutzten Programmiersprachen vorgestellt. Die Vorstellung ist keine Sprachreferenz sondern verdeutlicht grundlegend die Funktionsweisen und Aufgaben in der Präsentationsanwendung.

4.1 HTML

HTML steht für Hypertext Markup Language und ist durch das Internet zur wahrscheinlich meist verbreitesten Programmiersprache der Welt geworden. Definiert wird HTML durch die Standard Generalized Markup Language (SGML), welche der ISO-Norm 8879 unterliegt. HTML ist eine Auszeichnungssprache, das heißt Bestandteile eines Textdokumentes werden durch Zeichenfragmente mit bestimmten Eigenschaften versehen. Die Auszeichnung der Textelemente erfolgt im sogenannten *body* Abschnitt des HTML-Codes. Um ein HTML-Dokument ohne die Auszeichnungselemente betrachten zu können wird ein Interpreter benötigt. HTML-Interpreter sind zum Beispiel Browser, wie der Mozilla Firefox oder der Internet Explorer. Der Interpreter liest das HTML Dokument und entfernt für die Darstellung des Textes, die als *tags* bezeichneten Auszeichnungselemente. Diese werden analysiert, um die Textelemente in der gewünschten Formatierung darstellen zu können.

```
<Überschrift> Rostocker Fracht- und Fischereihafen <ENDE Überschrift>
```

Beispiel 1 Pseudocode für HTML Syntax

In den *tags* können auch dokumentenfremde Elemente geladen werden, zum Beispiel Karten oder Legenden. Der Aufbau der einzelnen Fremdobjekte spielt dabei für den Interpreter keine Rolle. Wichtig ist, dass das fremde Dateiformat vom Interpreter erkannt und als solches ausgegeben werden kann. Die meisten Browser benötigen unter anderem für das Flashformat *.swf ein Addon[9] oder ein Plugin[10].

In dem *body* vorangestellten Abschnitt *header*, werden Statusinformationen oder verborgene Programmteile festgehalten.

```
<img src="../tmp/rostock_karte.png">
<img src="../legenden/rostock_legende.png">
```

Beispiel 2 Import von Imageelementen

Die hier erwähnten Eigenschaften machen es möglich statische Karten über den Browser mit

[9] Erweiterung eines Programmes, durch Einbettung neuer Software
[10] siehe Addon

HTML zu erzeugen. Änderungen sind nur über das Neuladen des HTML-Dokumentes möglich.

4.2 PHP

Hinter der Abkürzung PHP steckt Hypertext Preprocessor. Es ist eine serverseitige Programmiersprache, die den Webseiten mehr Dynamik verleitet. Anders als bei HTML wird der PHP Code nicht erst vom Client gelesen und ausgewertet, dagegen erfolgt die Interpretation auf dem Server. Ein Serverinterpreter wandelt den PHP-Code in eine vom Browser interpretierbare Sprache um. Im allgemeinen Fall ist es eine Umwandlung in das HTML Format. Der veränderte Code wird als HTML-Seite vom Browser gelesen und ausgegeben. PHP bietet dabei die Möglichkeit Interaktionen mit Datenbanken auszuführen. Auf Grund der serverseitigen Umsetzung des Quellcodes werden dem Client nur die Daten übermittelt, die wirklich gebraucht werden. Somit gibt es keine Einsicht in die PHP-Programmstruktur oder die Datenbank. Im HTML-Dokument kann PHP an beliebiger Stelle eingebunden werden, es ist lediglich eine Kennzeichnung des Codes notwendig.

```
<html>
  <head>
        <title>In Rostock...</title>
  </head>
        <body>
              <?php
              echo "...kann man viel Schiffe sehen";
              ?>
        </body>
 </html>
```

Beispiel 3 Einfügen von PHP-Code in eine HTML-Webseite

4.3 JavaScript und AJAX

JavaScript ist eine clientseitige Programmiersprache, die Webseiten mit Dynamik erfüllen kann. Durch die Einführung von Web 2.0 und AJAX setzt sich JavaScript gegenüber den Sicherheitsbedenken durch. Das Einbinden von JavaScript-Programmcode in eine Webseite ist relativ einfach, ähnlich wie bei PHP müssen nur der Anfang und das Ende durch *tags*

markiert sein. Die Skriptfragmente können dabei auch im HTML Event-Listener[11] auftauchen. Diese dürfen jedoch keine Zeilenumbrüche enthalten, weil sie sonst nicht als Einheit vom Interpreter wahrgenommen werden.

Für die Interpretation von JavaScript im Facility Management System ist lediglich ein aktueller Webbrowser nötig. Denn anders als bei Java ist keine zusätzliche Laufzeitumgebung[12] notwendig.

```
<script type="text/JavaScript" src="jsfunktionen/suche.js"></script> //externe Funktion
<input type="submit" onclick="reset()" value="reset"> //Funktionsfragment
<script type="text/JavaScript"> //Einleitung JavaScript
        var map = createMap(); // JavaScript Programmcode
</script>//Ende JavaScript-Teil
```

Beispiel 4 Implementierung von JavaScriptcode in eine HTML-Webseite

Hinter der Abkürzung AJAX steht in diesem Falle kein Trojanischer Kriegsheld, sondern *asynchronous JavaScript and XML*[13]. Mit Hilfe von AJAX können während der Webseitennutzung Anfragen an den Server gestellt werden, ohne dass der Nutzer dieses bemerkt. Die Anfrageresultate werden vom Server im XML-Format zurückgeliefert. Der Browser tauscht nur die bestehenden DOM[14]-Inhalte gegen die aktuellen aus oder fügt die Ergebnisse in das bestehende DOM ein. Das bedeutet, dass bei der Interaktion des Webseitenbesuchers mit der Webseite nur die DOM-Elemente nachgeladen und ausgetauscht werden müssen, nicht aber die Seite kurzzeitig verschwindet und neuaufgebaut wird.

[11] JavaScript-Funktion die auf ein Ereignis, wie zum Beispiel einen Mausklick reagiert, und eine Aktion veranlasst
[12] Laufzeitumgebungen ermöglichen die plattfromunabhängige Ausführung von Programmcode
[13] Extensible Markup Language
[14] Document Object Model – Unterteilt ein Dokument in verschiedenen, austauschbare Objekte

Software

5 Server

Als Server werden lokale Rechnereinheiten bezeichnet, auf denen Serversoftwarepakete installiert sind. Eine solche Rechnereinheit ist die zentrale EDV-Schnittstelle des RFH-Intranets. Alle Arbeitsplätze die für das neue RFH Facility Management System relevant sind, sind mit diesem Server verbunden. Bei einer solchen Struktur wird von einer Server-Client-Struktur gesprochen. Der Server ist 24 Stunden im Einsatz und bearbeitet die Dienstanfragen der Clients, welche in diesem Fall die Arbeitsplätze mit Intranetzugang sind. Ein Dienst ist ein auf dem Server installiertes Softwarepaket, der zuständig für die Verarbeitung von Anfragen ist beziehungsweise die Regulierung des Datenaustauch zwischen Server und Client übernimmt. Unter anderem gewährleisten Serverdienste das Empfangen und Verschicken von eMails oder den Web-Zugriff. Im Folgenden werden die Softwarepakete mit ihren Funktionen beschrieben, die auf dem Server der RFH GmbH im Rahmen des Facility Management Systems installiert wurden.

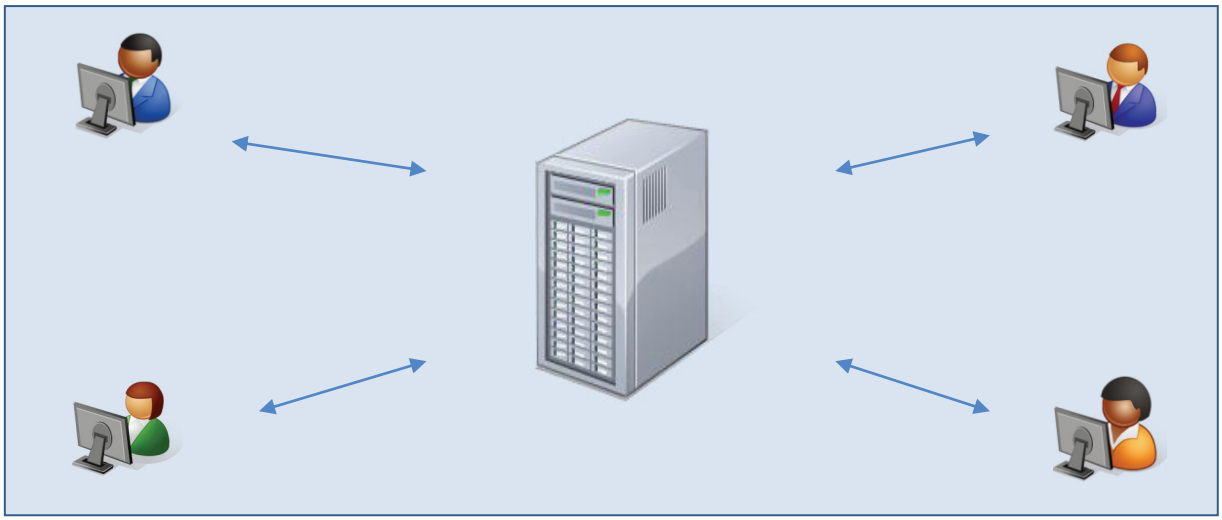


Abbildung 1 Server-Client-Modell

5.1 Webserver – Http Apache

Im vorhergehenden Abschnitt wurde beschrieben, dass Server Dienstprogramme auf ihrer Festplatte installiert haben. Der http Apache Webserver ist eine solche Dienstanwendung. Korduan und Zehner definieren Webserver als „Programme, die Inhalte im [Internet oder Intranet] bereitstellen und Anfragen von Web-Clients annehmen und abarbeiten können.“[14] Für die Informationsübertragung zwischen Webservern und Clients dienen die standardisierten Übertragungsprotokolle HTTP und HTTPS, sowie die Netzwerkprotokolle TCP/IP[15]. Laut selfhtml.org ist der am häufigsten eingesetzte Webserver der Apache HTTP-

[15] Transmission Control Protocol und Internet Protocol, ermöglichen den Datenaustausch

Webserver. Entsprungen ist der Apache der Unix-Welt. Seine starke Akzeptanz führte schnell zu verschiedenen Versionen für andere Betriebssysteme und Plattformen. Ein großes Plus des Apache Webservers ist sicherlich seine Open Source Lizensierung. Wodurch die Anschaffung auch für kommerzielle Anwendungen über das Apache Internetportal[16] ohne hohen Aufwand zu bewerkstelligen ist und keine Kosten entstehen lässt.

5.2 Apache Tomcat

Das Apache Tomcat Softwarepaket ist kostenfrei auf *http://tomcat.apache.org/* verfügbar und kann auf Grund seiner Open Source Lizenzierung ebenfalls unentgeltlich eingesetzt werden. Die Verwendung von Tomcat ermöglicht die Ausführung von Javacode auf dem Webserver. Tomcat ist ein Catalina-Servlet-Container, der durch den Jasper-JSP-Container zusätzlich JavaServer Pages in Servlets übersetzt und ausführt. Der komplett in Java gehaltene Quellcode ist auf der bereits genannten Homepage einsehbar. Damit die dynamische Ausführung von Java auf dem Webserver gewährleistet ist, wird der http Apache Webserver mit Hilfe des mod_jk Plugins erweitert. Dieses Plugin sorgt für die Weiterleitung der Requests an Tomcat.

5.3 Mapserver

Webserver sind nicht eigenständig in der Lage Maprequests zu verarbeiten und müssen die Karten- und Informationsanfragen an einen Mapserver weiterleiten. Mapserver lesen und verarbeiten dann die Anfragen. Nach erfolgter Auswertung übermittelt der Mapserver die Anfrageergebnisse an den Webserver, über den die Ausgabe an den Client erfolgt.
Im Folgenden werden der UMN Mapserver und der Geoserver näher vorgestellt.

5.3.1 UMN Mapserver

Die Entwicklung des UMN Mapservers wurde unter der Regie von Stephen Lime an der Universität von Minnesota ins Leben gerufen. Derzeit treibt das TerraSip-Projekt die Weiterentwicklung voran. Der UMN Mapserver ist unter mapserver.org frei zugänglich und kann auf Grund seiner GPL-Lizensierung auch kommerziell genutzt werden. Die kartenliefernde Software läuft auf vielen Betriebssystemen und ist mit „fast jedem beliebigen Webserver“[21] koppelbar.

[16] http://www.apache.de

5.3.1.1 Funktionsweise

Die Darstellung von Karten durch den Mapserver wird durch die drei Komponenten Mapfile, CGI-Formularvariablen und Ersetzungstexte erreicht. Im Mapfile stehen alle Informationen, die für die Kartendarstellung benötigt werden. Hier sind die darzustellenden Kartenelemente, die Datenherkunft und die Verwendung von Templates[17] durch den Mapserver konfiguriert. CGI[18]-Formularvariablen kontrollieren die Vorlageninhalte und ermöglichen Interaktionen des Anwendungsnutzers mit der Webanwendung. Die Ersetzungstexte sind in eckigen Klammern umfasste, reservierte Wörter, die sich in den Vorlagen befinden. Mapserver ersetzt diese durch die entsprechenden Inhalte. Ein temporärer Kartenlink tauscht zum Beispiel den Platzhalter [*img*] durch eine erzeugte Karte aus. Der Ablauf bis zur Kartendarstellung sieht dabei wie folgt aus: Der Webserver erhält die Clientanfrage und die dabei per POST oder GET übergebenen Variablen. Nachdem Erhalt werden diese in CGI-Variablen umgeschrieben. Im Anschluss wird das CGI-Programm mapserv(er.exe) aufgerufen, dieses ist für die Verarbeitung der Variablen zuständig. Als Resultat wird die temporär erzeugte Karte über den Webserver an den Client geliefert. Alternativ können die Variablen auch über ein Skript versendet werden. Diese Skripte werden als Mapscripts bezeichnet. Mapscripte gibt es für unterschiedliche Programmiersprachen, wie zum Beispiel phpMapScript für PHP und PerlMapScript für Perl.

Der Client muss über die Variable den Ort und die Bezeichnung der zu nutzenden Mapdatei übermitteln. Die Übergabe erfolgt über einen direkten Aufruf in der Dressleiste oder durch eine Definition im HTML-Code. Durch diese HTML-Seite können außer den Mapvariablen noch zusätzliche Variablen übergeben werden. Diese dienen zur Voreinstellungen und sind für den Nutzer durch HTML-Eigenschaften (*hidden*) meist nicht sichtbar. Wie bereits erwähnt werden diese Informationen an den Mapserver weitergeleitet. Dieser liest das Mapfile Zeile für Zeile ein und kontrolliert die Syntax. Entspricht diese der Mapfilespezifikation, werden die Informationen verarbeitet. Die Karten und das Kartenzubehör, wie Maßstabsbalken oder Legenden, werden nach der Auswertung der Mapdatei generiert. Die Daten werden vom Mapserver zum Beispiel aus Geodatenbanken oder GIS typischen Dateien gewonnen. Nach der Erzeugung, der vom Client angefragten Karten, werden diese im temporären Mapserververzeichniss gespeichert und bereitgestellt. Nun werden die Templates gelesen und deren Platzhalter durch die erzeugten Daten ausgetauscht. Für die Karten werden die Links zu den temporären Imagepfaden gesetzt. Die weiteren Informationen werden in Form von

[17] Vorlagen die während der Nutzung an die aktuellen Gegebenheiten angepasst werden

[18] Common Gateway Interface – dient als Schnittstelle zwischen einem webserver und zusätzlichen Softwarepakten

Variablen neu definiert. Um aus denen im Mapfile festgelegten Vorlagen eine einheitliche HTML-Seite zu gestalten, müssen strikte Regeln eingehalten werden. Diese Regeln sind auf der Homepage[19] des Mapservers dokumentiert.

Die Sachdatenabfrage erfordert eine eigene Struktur der ausgebenden Vorlagen. Die HTML-Vorlagen werden dabei für jeden abgefragten Datensatz ausgefüllt und aneinander gehängt. Spezielle Funktionen ermöglichen die einmalige Ausgabe bestimmter Vorlagenabschnitte. Es ist dabei anzumerken, dass Geometriedaten auch bei einer Sachdatenabfrage mit ausgegeben werden können.

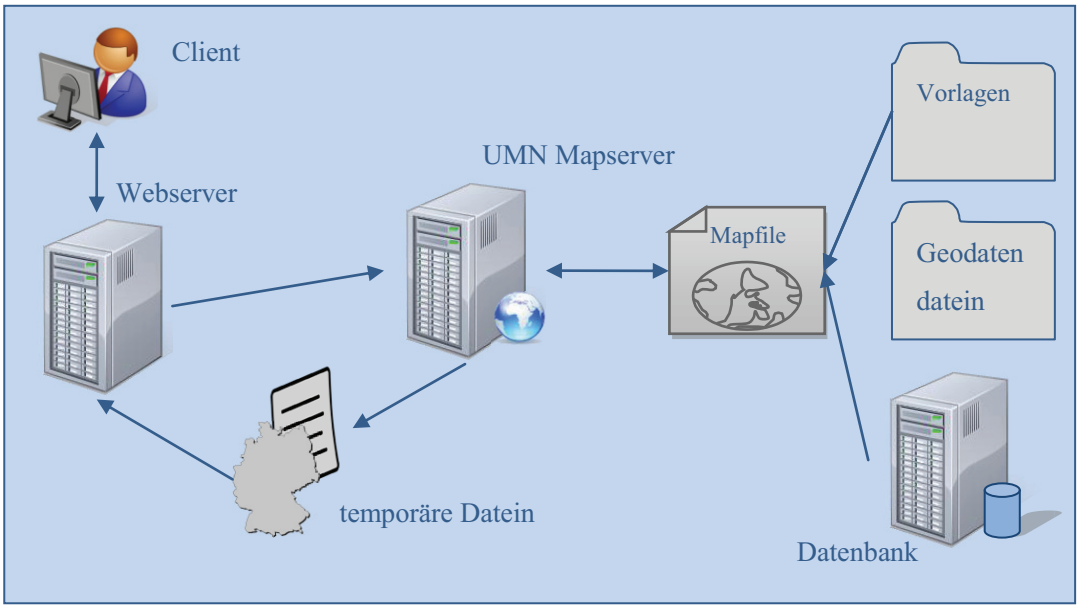


Abbildung 2 Funktionsablauf UMN Mapserver vgl. [14 s.194]

Bei der Programmierung von Templates gilt es zu bedenken, dass die Quelltexte nur vom Mapserver verarbeitet werden. Das heißt der Webserver übergibt die verarbeiteten Dateien ohne eine erneute Bearbeitung an den Client. Für die Programmierung gilt deswegen, dass serverseitige Programmiersprachen wie PHP nicht mehr ausgeführt werden. Aus diesem Grund sollten clientseitige Programmiersprachen verwendet werden, weil diese erst clientseitig vom Browser interpretiert werden.

Eine echte Grenze in der Anwendung von Mapserver ist die dynamische Gestaltung von Templates die auf dem Server in Wechselwirkung mit Datenbankinhalten generiert werden sollen. Derzeit ist eine solche Möglichkeit nur über die Brücke der clientseitigen Umsetzung gegeben. Korduan und Zehner gehen aber davon aus das „ein AJAX-Client für CGI-MapServer […] nicht mehr lange auf sich warten“ [14] lässt.

[19] http://mapserver.org

5.3.1.2 Mapfile

Im Mapfile sind alle notwendigen Parameter festgehalten, um Geodaten über den Mapserver auszugeben. Das Konzept ist denkbar einfach, Objekte werden durch Schlüsselwörter eingeleitet und mit dem Wort *END* geschlossen. Zwischen dem Schlüsselwort und *END* können, neben den Definitionen, auch Unterobjekte festgelegt werden. Die einzeiligen Definitionen unterliegen dem Keyword-Value-Pair (KVP) Prinzip. Dabei folgt auf ein Schlüsselwort, durch einfache Leerzeichenabtrennung, eine Wertzuweisung.

```
NAME  GelaendeRFH      #Schlüsselwort und Wert
```

Beispiel 5 Key-Value-Pair

Schlüsselwörter können innerhalb des Mapfiles häufiger verwendet werden. Die Bedeutung ist vom zugeteilten Objekt abhängig. Das Schlüsselwort NAME kann z.B. einem Layer ebenfalls der Gesamtkarte einen Wert zuweisen.

Werte die die Eigenschaften eines Objektes näher beschreiben:

- beliebige Texte, z.B. für die Namensgebung NAME Karte
- vorgegebene Texte, z.B. Ausgabeformat MIMETYPE "image/png"
- festgelegte Konstante, zB. beim Postitionsbezug der Texte POSITION [ul|uc|ur|cl|cc|cr|ll|lc|lr]
- Zahlen und Zahlenfolgen, z.B. EXTENT 4505385 5998305 4506040 5999781
- Wahrheitswerte, z.B. STATUS [on|off|*default*]
- Ausdrücke die durch logische Operatoren verbunden werden, z.B. bei FILTER „ ‘[Nutzung]‘ eq ‘Alter Hafen‘“
- Pfadangaben, z.B. SYMBOLSET "../symbols/symbols.sym"

Ausnahmen in der Wertzuweisungsformation kommen in der Objektkategorie PROJECTION vor, hier bilden Schlüsselwörter und Werte einen Wert z.B. "init=epsg:2398".

Das Herzstück des Mapfiles ist das Layerobjekt, in dem die Beschreibung der Ausgangsdaten und die optischen Ausgabeparameter für die Ausgabe festgelegt werden. Das Schlüsselwort LAYER leitet ein solches Objekt ein. Ein Layerobjekt muss innerhalb seiner Beschreibung typiesiert werden, weil Mapserver keine unterschiedlichen Geometrien in einem Layer darstellen kann. Zu den Geometrietypen zählen:

- Punkt
- Linie
- Polygon
- Annotation

- Raster
- Abfrage

Die Ausgabe von komplexeren Geometrien wird durch Layerabhängigkeiten erzeugt, das heißt Layer A kann nicht ohne Layer B angezeigt werden. Diese Art der Abhängigkeit wird durch das Wort REQUIRE bestimmt.

Weiterhin wird in der Layerdefinition die Herkunft der geometrischen Daten angegeben. Als Datenquelle dienen:

- Dateien im Shapeformat
- ORG unterstützende Vektordatein
- Vektordatein die mit Hilfe von gdaltindex in einer Shapedatei zusammengefügt werden
- Geometrien aus Datenbanken
- Vektordaten aus einem Web-Feature_Service kommend
- Rasterdaten aus einem Web- Map-Service kommend
- von GDAL unterstütztes Rasterdatenformat
- TIFF-Dateien=>je Layer eine TIFF Datei
- mehrere, durch Tileindex zusammengefasste Rasterdateien

Die Datenart wird im Mapfile durch Schlüsselwort DATA und dem dazugehörigen Wert bestimmt. Mit CONNECTIONTYPE wird der Datenzugriff mit den benötigen Parameter definiert.

```
LAYER
  NAME raeume_gesamt
  CONNECTIONTYPE postgis
  CONNECTION "host=localhost dbname=rfh user=postgres password=postgres"
  DATA "the_geom from (select oid,* from raeume where the_geom is not null)"
  TYPE POLYGON
  STATUS ON
  METADATA
    "WMS_TITLE" "gebaeude"
    "WMS_SRS" "epsg:2398"
    "WMS_INCLUDE_ITEMS" "all"
  END
  PROJECTION
    "init=epsg:2398"
  END
  CLASS
    NAME "Ansiedlung"
    TEMPLATE getfeatureinfo.html
  END
END
```

Beispiel 6 Layerdefinition im Mapfile

Alle Parameter sind in der Mapfile-Referenz ausführlich und mehrsprachig erläutert.

5.3.2 Geoserver

Der Geoserver ist ebenfalls auf Grund seiner Open Source-Lizenz frei verfügbar und kann für Windows- und Macintoshrechner heruntergeladen[20] werden. Anders als der UMN Mapserver basiert der Geoserver auf Java und nicht auf C++. Der Vorteil des Geoservers liegt in seiner deutschsprachigen und graphischen Benutzeroberfläche durch die er komplett ohne zusätzliche Mapfiles auskommt. Das OGC hat den GeoServer zertifiziert, weil er die OGC-Standards für WMS, WFS und WCS-Dienste unterstützt.

5.3.2.1 Funktionsweise

Der GeoServer arbeitet auf Grundlage eines Request-Response-Subsystems, das heißt ein Dispatcher empfängt einen Request und wertet diesen aus. Diese Ergebnisse werden an die

[20] http://www.geoserver.org

zuständigen Programmteile (Operation) des GeoServers geschickt. Ein sogenannter AbstractService erhält die speziell für ihn wichtigen Parameter und konfiguriert einen RequestReader, um die Anfrage korrekt ausführen zu können. Der RequestReader liest die Anfrage und die dazugehörigen Parameter erneut ein. Mit Java-Bean-Code wird ein einzelnes Request-Objekt angelegt. Mit dem auf diese Art erzeugten Objekt, wird die Ausgabe angefordert. Am Ende der Funktionskette steht der Response, welcher die eindeutig parametrisierte Anfrage ausführt und das Ergebnis ausliefert.

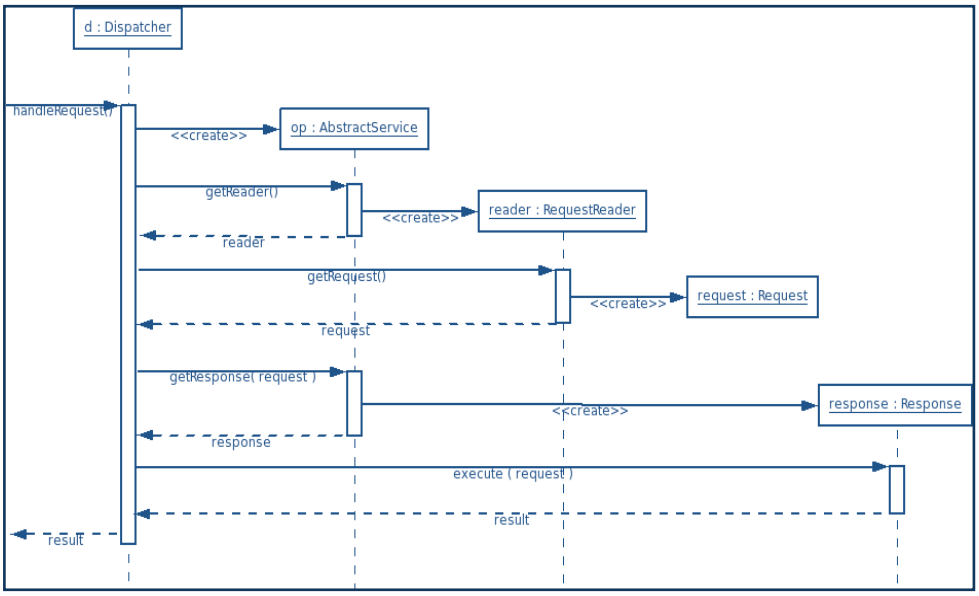


Abbildung 3 Funktionsweise Geoserver [9]

In Abbildung 4 wird ein WFS GetCapabilities-Request an den GeoServer geschickt. Der Dispatcher wertet diesen Request aus und leitet die nötigen Parameter weiter. Das Teilprogramm, welches für die Bearbeitung von GetCapabilities-Anfragen der WFS-Dienste zuständig ist, erhält die Parameter und wertet diese aus. Auf Grundlage der Auswertung wird nun ein Javacode generiert, der für die Anfrage steht. Der Response führt den Javacode aus und erzeugt das GetCapabilities-Dokument und gibt dieses aus.

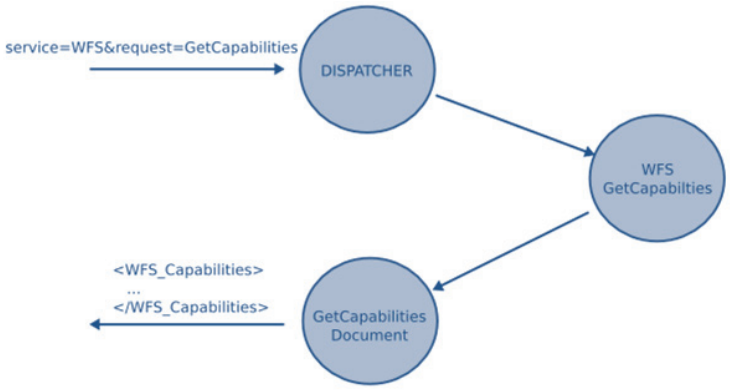


Abbildung 4 Beispiel eines GetCapabilites-Request mit Geoserver [9]

5.3.2.2 Verwendung der Geoserver Oberfläche

Die Benutzeroberfläche vom GeoServer ersetzt das Schreiben eines Mapfiles, wie es beim UMN Mapserver verlangt wird. Die Vorgehensweise ist sowohl für WFS und WMS als auch dem WCS gleich. Bei der ersten Nutzung von GeoServer wird der Administrator konfiguriert, dabei reicht die Festlegung eines Nutzernamens und Passwortes.

Als ersten Schritt, auf dem Weg zur Einrichtung eines Dienstes, wird ein Namensraum eingerichtet. Der Namensraum dient dazu, dass mehrere Projekte mit identischer Datenquelle unterschieden werden und in der Datenbankanfrage eindeutig angesprochen werden können.

Abbildung 5 Konfiguration des Namensraum

Im DatasStore Menü kann dem Namensraum eine Datenquellen zugeordnet werden. Es können Shapefiles, PostGIS-Datenbanken oder externe WFS-Dienste als Grundlage für die Ausgangsdaten eingerichtet werden. Bei der Zuordnung einer Datenquelle müssen deren Ort und die Zugangsberechtigungsdaten angegeben werden. Sollte bei einer PostGIS-Datenbank nicht the_geom der Geometriespaltenname sein, muss auch dieses exakt benannt werden.

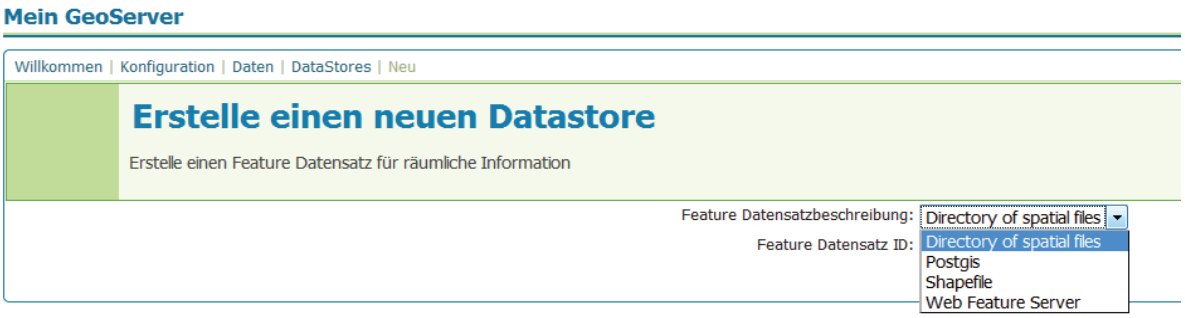


Abbildung 6 Erzeugen eines Datastores

Im nächsten Schritt werden den Geometrien ihre Eigenschaften verliehen. Im Konfigurationsmenü FeatureTypes wird eine Projekt und die gewünschte Datenquelle ausgewählt. Nach der Auswahl werden der Geometrietyp, die Projektion und die Bounding Box festgelegt. Zusätzlich können noch Metadaten angegeben werden. Die gesamten Metadaten werden für jeden einzelnen Service in einem gesonderten Menüpunkt eingetragen.

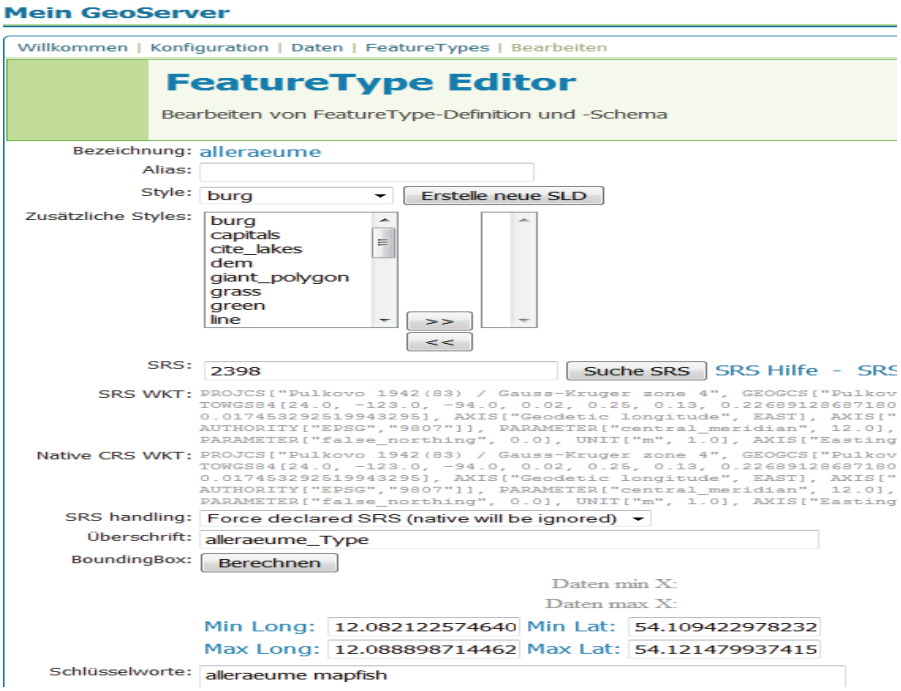


Abbildung 7 Eigenschaften des Features festlegen

Ob die Konfiguration des Dienstes den eigenen Wünschen entspricht, kann auf der Demo-Vorschauseite begutachtet werden. Um nun den Dienst in die eigene Applikation einzubinden, muss eine Anfrage an den Geoserver gestellt werden. Dieser kann über „get“ oder „post“ erfolgen. Bei der Post-Methode wird die Anfrage in dem Körper des Antragsdokumentes angegeben. Eine Post-Anfrage wird zumeist durch ein XML-Dokument gestellt.

Methode	Anfrage
Get	<protocol>://<host>:[port]/geoserver/<service>?request=<request>
Get	<protocol>://<host>:[port]/geoserver/<service>/<request>
Get	<protocol>://<host>:[port]/geoserver/ows?Service<service>&request=<request>
Post	<protocol>://<host>:[port]/geoserver/service

Tabelle 1 Anfragesyntax für Post und Get

6 Client

Im Facility Management System des RFHs ist der Client ein Arbeitsplatz mit Zugang zum hausinternen Intranet. Um die Dienste der Präsentationsplattform nutzen zu können, ist lediglich ein Browser notwendig. Für die Verwaltungszwecke wird eine clientseitige Installation von MS Access für die Sachdatenverwaltung und ein Desktop GIS für die Verwaltung von Geometriedaten benötigt.

6.1 Webbrowser

Webbrowser, allgemein als Browser bezeichnet, dienen der visualisierten Ausgabe von Daten und Dokumenten speziell aus dem Internet. Ursprünglich als Benutzeroberfläche für Webseiten gedacht, bieten moderne Browser (z.B. Firefox 3.5), durch die clientseitige Interpretation von JavaScript die Möglichkeit dynamische Webseiten im Stile von Web 2.0 darzustellen. Die Implementierungsunterschiede zwischen den einzelnen Browsern führen zu erheblichen Unterschieden der Browserleistungen. Im Folgenden werden der Marktführende Browser Internet Explorer von Microsoft (Marktanteil 67,68%[11]) in der Version 8, der ewige Konkurrent Mozilla Firefox (22,47%) in der Version 3.5 und der neue Herausforderer Google Chrome (2,59%) in der Version 4.0 vorgestellt. Die richtige Wahl des Browser ist für das Facility Management System innerhalb der RFH GmbH von großer Bedeutung, denn der Browser soll als Präsentationsplattform dienen und dabei die Daten möglichst zuverlässig und schnell verarbeiten und ausgeben können. Bei der Verwendung des Browsers als Präsentationsoberfläche darf dieser für den RFH kein zusätzliches Sicherheitsrisiko darstellen. Um den Leistungsumfang der Browser besser einschätzen zu können, werden die Acid[21]-Tests im Vorfeld kurz erklärt um das Verständnis der einzelnen Werte in der Browservorstellung zu erhöhen.

6.1.1 Browsertests

Mit Hilfe der Acid-Tests wird der Funktionsumfang eines Browser festgestellt. Die derzeit aussagekräftigsten Tests sind die vom WaSP[22] entwickelten Acid2 und Acid3 Tests. Der Acid2-Test prüft die Browser auf Render[23]-Fehler bei der Darstellung von Webseiten. Die Webbrowser werden auf die Einhaltung der HTML Prinzipien und der Unterstützung von CSS untersucht. Browser die durch ihre Implementierungen von HTML und CSS[24] die

[21] Eigenname der Tests
[22] Web Standards Project
[23] Umsetzung von Programmcode in eine, am Bildschirm, dargestellte Webseite
[24] Cascading Stylesheets, dienen der exakten Postionierung und Formatierung von HTML-Objekten

Richtlinien der W3C Spezifikation unterstützen, haben bei der Darstellung der getesteten Bilder keine Probleme und bestehen den Test.
Der ACID3-Test beschäftigt sich vorwiegend mit der Prüfung auf Kompatibilität der Browser mit interaktiven Webseiten. Dabei werden die Browser hauptsächlich auf die Unterstützung von DOM-Level 2 und dem ECMA Script geprüft. Aber auch eine Prüfung der Darstellung von SVGs findet statt.
Um die beiden Tests zu bestehen müssen die Kriterien jeweils zu 100% erfüllt sein.

6.1.2 Internet Explorer

Der Marktführer unter den Webbrowsern ist Mircosofts Internet Explorer. Der aktuell in der Version 8 verfügbar ist. Im Gegensatz zu seinen Vorgängern hat die Version 8 ein komplett überarbeitetes Sicherheitssystem. Der Smartfilter schützt den Anwender vor dem Besuch von schädlichen Webauftritten. Die Anonymität des Internetsurfers gegenüber Unternehmen die ihre Webseiten zu Spionagezwecken missbrauchen ist nach Angaben von Microsoft gewährleistet. Ein weiteres großes Plus, neben der bereits vorhandenen Installation auf Windowssystemen, ist die optionale Einstellung von Sicherheitszonen. Mit dem einfachen Festlegen von fünf verschiedenen Sicherheitszonen ist der Internet Explorer für administrative Netzwerke geeignet. Die Erhöhung der Sicherheitszone bedeutet jedoch auch ein Verzicht auf JavaScript, ActiveX[25] und Cookies[26], wodurch viele Webpräsentationen nicht oder nur fehlerhaft angezeigt werden können.
Mit der Einführung des Internet Explorers 8 besteht Microsoft auch erstmals den Acid 2-Test. Jenes liegt einfach daran, dass die achte Version des Explorers, im Vergleich zu älteren Versionen, nahe am W3C[27]-Standard festhält und endlich auch CSS2.1 unterstützt. Nachteilig an diesem Erfolg ist, dass viele Webauftritte an die nicht Einhaltung des W3C-Standards von Vorgängerversionen angepasst wurden und nun fehlerhaft dargestellt werden.
Ein weiteres Manko ist, dass das von Microsoft als schnellster Browser der Welt angepriesene Programm starke Performanceeinbrüche bei der Arbeit mit dynamischen Webseiten hat. In den aktuellen, von ZDNet durchgeführten Benchmarks, die Browser auf Geschwindigkeit und Performance bei der Nutzung von Web 2.0 Auftritten testen, liegt der Internet Explorer in allen Bereichen und vor allem auch in jedem Test auf den letzten Rängen.
Auch der Acid 3-Test gibt keinen Grund zum Jubel, denn der Internet Explorer erreicht gerade einmal 20 von 100 möglichen Punkten.

[25] ermöglicht die Ausführung aktiver Software (z.B. Videos) innerhalb einer Webseite
[26] erlauben einer Anwendung Information auf dem Computer zu hinterlegen
[27] World Wide Web Consortium

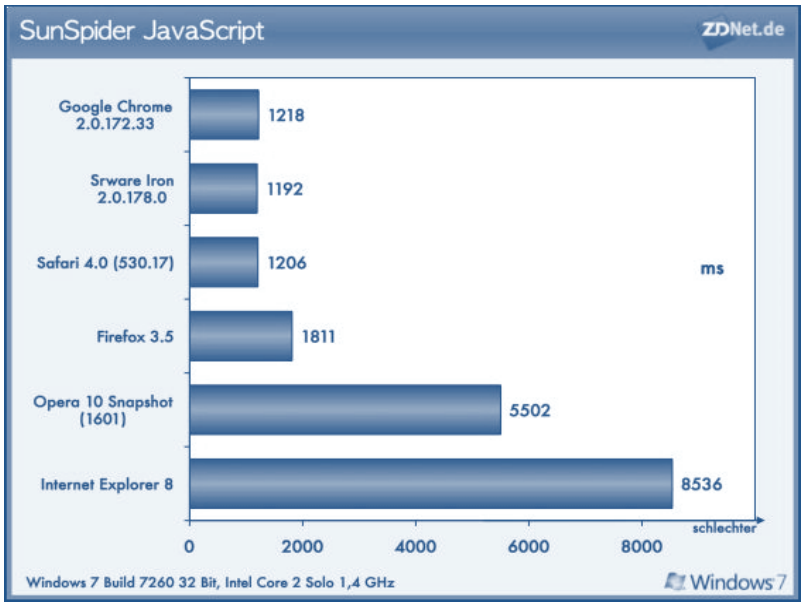


Abbildung 8 Internet Explorer mit schlechtem SunSpider-Testergebnis [2]

6.1.3 Mozilla Firefox

Nachdem Untergang von Netscape, gründeten ehemalige Entwickler das Mozillaprojekt um den einstigen Vorzeigebrowser wieder in die Spur zu führen. Noch während des Scheiterns wurde unter derselben Regie ein weiteres Browserprojekt gestartet, Firefox war geboren. Aktuell ist der Firefox in der Version 3.5 (Stand August 2009) verfügbar. Stetig steigende Marktanteile lassen ihn zum Hauptkonkurrenten des Internet Explorers werden.

Mit der Einführung der 3.5 Version wurden nicht nur 4 Versionsnummern übersprungen, sondern laut Tristan Nitot[28] auch 5000 Verbesserungen, Änderungen und neue Funktionen zur Vorgängerversion (Version 3.0) integriert. Ähnlich wie Microsoft setzt auch Mozilla auf ein ausgeklügeltes Sicherheitspaket. So bietet Firefox schutz vor Malware[29] und einen Phishing[30]-Filter. Für Webseiten, die nicht standardisierte Sicherheitszertifikate nutzen, können im Browser Ausnahmen festgelegt werden. Diese dienen dazu, dass vertrauenswürdige Seiten weiterhin besucht werden können beziehungsweise böswillig programmierte Webseiten vom Nutzer erkannt werden und so keinen Schaden anrichten. Firefox bietet zudem eine Funktion, die sich private browsing nennt, damit kann der Nutzer seine Spuren verwischen und niemand kann erkenne welche Seiten besucht worden sind. Ein weiteres Sicherheitsmerkmal bietet das Zusammenspiel zwischen Firefox und der lokal installierten Antivirensoftware. Dieses Zusammenspiel ermöglicht das Prüfen von Dateien nach Viren und Würmern noch während des Datendownloads.

[28] Präsident vonMozilla Europa
[29] Computerprogramme, die unwissentlich installiert werden und Schaden auf dem Rechner ausführen
[30] Versuche über gefälschte Internetardressen an Daten des Internetnutzers zu gelangen

Neben der hohen Sicherheit spricht der hohe Leistungsfaktor für Mozillas Firefox. Durch die Implementierung von HTML 5 Standards ist es nicht mehr zwingend nötig multimediaformatunterstützende Plugins zu installieren. Durch eine eigene Speicherverwaltung lässt Firefox Systemprozesse nahezu unberührt und bewahrt somit die Performance des gesamten Systems. Im Vergleich zur Version 3.0 führt die komplette Überarbeitung der JavaScriptengine *TraceMonkey* zu einer Verdopplung der JavaScript Ausführungsgeschwindigkeit im *SunSpider* Benchmark[31]. Im ZDNet-Ranking landet der Firefox mit diesen Werten im gehobenen Mittelfeld und setzt sich dadurch deutlich vor den Internet Explorer. Auch in den Acid-Tests ist der Firefox vor dem Konkurrenzprodukt Microsofts, bereits die Vorgängerversionen hatten den Acid2 Test erfolgreich bestanden und auch im Acid3- Test wurden immerhin 93 von 100 Punkten erzielt.

Ein weiteres Plus von Mozillas Webbrowser ist der offengelegte Quellcode. Dieses ermöglicht Erweiterungen durch eigene und von der großen Community[32] zur Verfügung gestellter Addons. Diese Addons können den Funktionsumfang erheblich steigern, zum Beispiel können mit dem *Firebug*-Addon CSS, HTML und JavaScript während des Webseitenbesuchs geändert, auf Programmfehler untersucht und überwacht werden. Zusammenfassend kann gesagt werden, dass der Firefox mehr als eine einfache Alternative zum Internet Explorer darstellt.

6.1.4 Google´s Chrome

Am 2.September 2008 ist Google mit Chrom offiziell in den Browserwettstreit eingestiegen. Google selbst stuft Chrome als OpenSource-Browser ein, weil der Quelltext auf dem firmeneigenen Open Source-Projekt Chromium aufsetzt. Für das Rendern von Webseiten ist die freie HTML-Rendering-Bibliothek WebKit verantwortlich, diese wird bereits in Apples Safaribrowser erfolgreich verwendet.

Wie die beiden bereits erwähnten Browser setzt auch Google auf ein hohes Maß an Sicherheit. Phishing-Filter, Malwareschutz und die Überprüfung von rückgelieferten Sicherheitszertifikaten sollen den Computer vor den Gefahren des Internets schützen. Der Modus anonym browsen ermöglicht inkognitives surfen auf Webseiten. Dem Kritikpunkt der Spionage durch Google baut Google eigenständig vor, indem alle datenübermittelnden Dienste von Chrom ausgeschaltet werden können. Ob diese swirklich immer sinnvoll ist, muss jeder Nutzer eigenständig für sich beantworten. Denn regelmäßig überarbeitete Listen von schädlichen Webseiten werden automatisch zwischen Chrom und den Googleservern

[31] Test die es ermöglichen Soft- und Hardware objektiv zu vergleichen
[32] Zusammenschluss von Anwendern, die sich gegenseitig unterstützen

synchronisiert. Zudem versichert Google in den Datenschutzerklärungen, dass die Daten lediglich der Verbesserung von Googleprogrammen dienen. Was der Googlekonzern jedoch unter Verbesserungen versteht ist nicht erläutert.

Mit der Leistung von Chrome scheint Google vor allem auf Windowsrechnern neue Maßstäbe zu setzen. Die enormen Geschwindigkeiten, die Chrom bei der Verarbeitung von dynamischen Webseiten erzielt, ist sicherlich der neue JavaScriptengine *V8* zu verdanken. Ein weiterer geschwindigkeitsfördernder Punkt ist die Trennung von Programmprozessen und der Optik. Mit dieser Trennung macht Google es möglich, das bei Prozessabstürzen, nicht das ganze Programm neu gestartet werden muss, sondern nur der Tab in dem die Webseite ausgeführt wurde. Ein eigenständiger Task-Manager[33] ermöglicht zusätzlich die manuelle Verwaltung der Programmprozesse. So kommt es das Google neben dem Safaribrowser ein weiterer Browser ist, der den Acid3-Test besteht.

Ähnlich wie beim Firefox lässt sich der Funktionsumfang von Chrome durch die Einbindung von Addons erweitern. Mit dem Inspector ist jedoch schon ein Debugger[34] in Chrom implementiert.

[33] Programm zur Anzeige und verwaltung von Systemprozessen
[34] Programm zum Erkennen und Beheben von Fehlern im Programmcode

7 Datenbankmanagementsysteme

Das Facility Management System kommt nicht ohne eine zentrale Datenbank zur Verwaltung der Sach- und Geometriedaten aus. In diesem Abschnitt werden die beiden Datenbank Systeme MS Access und Postgresql vorgestellt.

7.1 Microsoft Access

MS Access ist ein proprietäres Datenbanksystem des Softwareriesen Microsoft. Microsofts aktuelles Datenbanksystem MS Access 2007 unterstützt, im Gegensatz zu vorhergehenden Versionen, das relationale Datenbankmodell. Microsoft hat bei der Programmierung von Access das Datenbankformat *.mdb eingeführt, das für die Speicherung von sämtlichen Daten innerhalb der Datenbank zuständig ist. Mittlerweile ist eine Trennung von Tabellendefinitionen (inklusive Daten) und der optischen Oberfläche möglich. Es wird zwischen *frontend* für die Oberflächenstruktur und *backend* für die Definition und Daten unterschieden.

MS Access dient lediglich der Speicherung und Verwaltung von „herkömmlichen“ Daten, eine Einbindung von Geometriedaten ist ohne zusätzliche Software nicht vorgesehen. Geometrieerweiternde Software für das Microsoft Datenbanksystem würde die RFH GmbH Geld kosten und zumeist nicht ohne eigene GIS-Software auskommen (GC Access Manager und Smallworld). Im Budget der RFH GmbH sind keine extra Kosten für die Präsentationsplattform einkalkuliert worden, daher wird nicht auf diese Extension eingegangen. Eine Verbindung zwischen Sachdaten und Geometriedaten ist aber zwingend erforderlich, deswegen ist die Verbindung zu externen Datenbeständen notwendig. MS Access bietet die Möglichkeit mit Datenbeständen anderer Datenquellen zu kommunizieren. Eine Variante ist die Verwendung eines ODBC-Treibers (siehe Kapitel 6.3).

7.2 Postgresql

Postgresql ist ein objektrelationales Datenbankmanagementsystem. Das Open Source Datenbanksystem kann kostenfrei heruntergeladen[35] werden. Der lizenzlose Einsatz erlaubt die Verwendung von Postgresql im kommerziellen Bereich. Den aktuellen Namen Postgresql besitzt, dass aus einem universitären Projekt hervorgegangen Datenbanksystem, seit 1996. Entstanden ist Postgresql aus einer Entwicklung des Datenbankmanagementsystems POSTGRES von der Universität „of California at Berkley“. Aktuell ist es in der Version 8.4

[35] http://www.postgresql.org

verfügbar. Seit der Veröffentlichung wird der offengelegte Code von Programmierern auf dem gesamten Globus modifiziert und verbessert.

7.2.1 Kommunikation

Der Arbeitsablauf von Postgresql beruht dabei auf dem Prinzip eines Client-Server-Modells, das heißt es findet ein ständiger Austausch zwischen Server und Client statt. Die Kommunikation zwischen dem Nutzer und dem Datenbankprogramm Postgresql übernimmt das Serverprogramm *postmaster*. Die Verbindungen und Datenbankdateien werden vom *postmaster* überwacht und verwaltet. Die Abarbeitung der Clientabfragen wird vom gleichen Serverprozess durchgeführt. Unterschiedlichste Clientprogramme ermöglichen die Kommunikation mit dem *postmaster*. Bei Ansteuerung von Webseiten übernimmt jedoch der Webserver die Rolle des Client.

Im Standardinstallationspaket befindet sich der pgAdminIII als graphische Verwaltungsoberfläche. Auf Grund des Client-Server Modells brauchen Client und Server nicht auf einem gemeinsamen Rechnersystem installiert sein. Die Kommunikation zwischen den beiden Einheiten erfolgt über TCP/IP Verbindungen.

7.2.2 Funktionsumfang

Der große Funktionsumfang ist ein Plus von Postgresql. Neben der Fähigkeit nicht atomare Daten[36] zu speichern, beherrscht Postgresql auch die Vererbung von Objektidentitäten. Der Nutzer kann innerhalb seiner Datenbank Datentypen, Operatoren und Funktionen an seine eigenen Ansprüche anpassen und gegebenfalls auch neu definieren. Gesteigert wird die Leistung von Postgresql durch die Unterstützung referentieller Integrität[37] und das Vorhandensein eines modernen Transaktionsmanagements. Desweiteren werden Definitionen von Triggern[38] und Regeln unterstützt, mit deren Hilfe Zugriffe auf Datenbankobjekte gesteuert werden. Der SQL92-Standard und der SQL99-Standard werden von Postgresql-Datenbanken unterstützt. Mit einem solchen Leistungsumfang kann die Herstellerthese vom „fortschrittlichste[n] Open-Source-Datenbanksystem[s]“ wohl gerechtfertigt sein, doch Postgresql kann durch eine Menge Addon´s noch zusätzlich an Funktionalitäten gewinnen.

7.2.3 PostGIS

PostGIS ist ein optionales Addon, welches den räumlichen Funktionsumfang von Postgresql erweitert. Die Implementierung der Simple Feature Access Spezifikation des OGC

[36] atomare Daten: d.h. einem Attribut dürfen nicht mehrere Werte eines Gültigkeitsbereiches zugewiesen werden
[37] Bedingungen zwischen Prozessen auf Beziehungsebene
[38] werden zur Wahrung der Datenkonsistenz während der Änderung von Datensätzen in Datenbanken verwendet

gewährleistet den Einsatz von PostGIS mit vielen weiteren standardisierten GIS-Programmen. Neben den „einfachen“ Geometrietypen Point, Linestring, Polygon, Multipoint, Multilinestring, Multipolygon und Geometriecollections im Well-Known-Text[39] oder Well-Known-Binary[40]-Format lassen sich die Geometrien durch das Extended WKT und das Extended WKB-Format um eine dritte Dimension, wie zum Beispiel Messwerten oder Höhen erweitern. Neben der Erstellung von Geometrien, ist die Verwendung vieler räumlichen Funktionen möglich. Es lassen sich Flächen- und Distanzmessungen durchführen, Geometrien verschneiden oder Pufferzonen berechnen. Mit Hilfe von PostGIS können die Geometrien in Beziehungen gesetzt werden, wodurch Abfragen wie Contains, Within oder Overlaps ermöglicht werden. PostGIS unterstützt unter anderem Abfrageformate WKB und WKT, sowie GML[41] und KML[42].

```
INSERT INTO gebaeude VALUES ('RFH', GeometryFromText ( 'POLYGON ( ( 75 20, 80 30, 90 22, 85 10, 75 20 ) )', 4326 ) );
```

Beispiel 7 SQL-Statement zur Importierung einer Geometrie

Der Zugriff auf die Geometriedaten wird durch das bereits erwähnte Verwaltungstool pgAdminIII im gesamten Postgresql-Kontext gewährleistet.

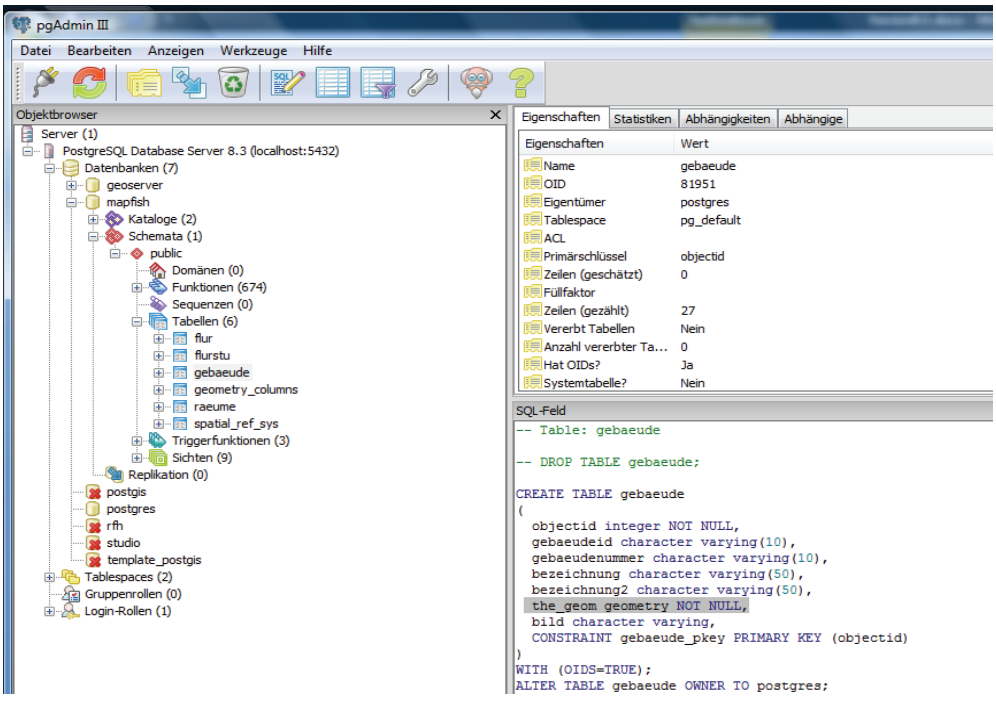


Abbildung 9 Oberfläche des pgAdminIII-Verwaltungstools

[39] WKT, lesbare Form der Datenspeicherung z.B. LINESTRING(3 4,10 50,20 25)
[40] WKB, binäre Form der Datenspeicherung z.B. Punkt (1,1) ist 010100000000000000000F03F000000000000F03F
[41] Geography Markup Language, Anwendung von XML zum Austausch raumbezogener Daten
[42] Keyhole Markup Language, Anwendung von XML speziell zur Anzeige räumlicher Daten in Google Earth

Neben einer allumfassenden Dokumentation, ist wird auch der kostenfreie Download auf der Homepage[43] vom Entwicklerteam Refractions bereitgestellt.

7.3 ODBC

Open Database Connectivity (ODBC) ist eine, unter Verwendung von SQL als Datenbanksprache, standardisierte Datenbankschnittstelle. Diese ermöglicht die Kommunikation zwischen verschiedenen Datenverwaltungsprogrammen. Der ODBC-Treiber ermöglicht es, im neuen Facility Management System der RFH GmbH die bestehende MS Access Datenbank als Verwaltungsprogramm beizubehalten, wobei die Daten im Hintergrund auf einer Postgresql-Datenbank hinterlegt werden. Mit dieser Möglichkeit wird gewährleistet das Sachdaten nicht redundant gespeichert werden.

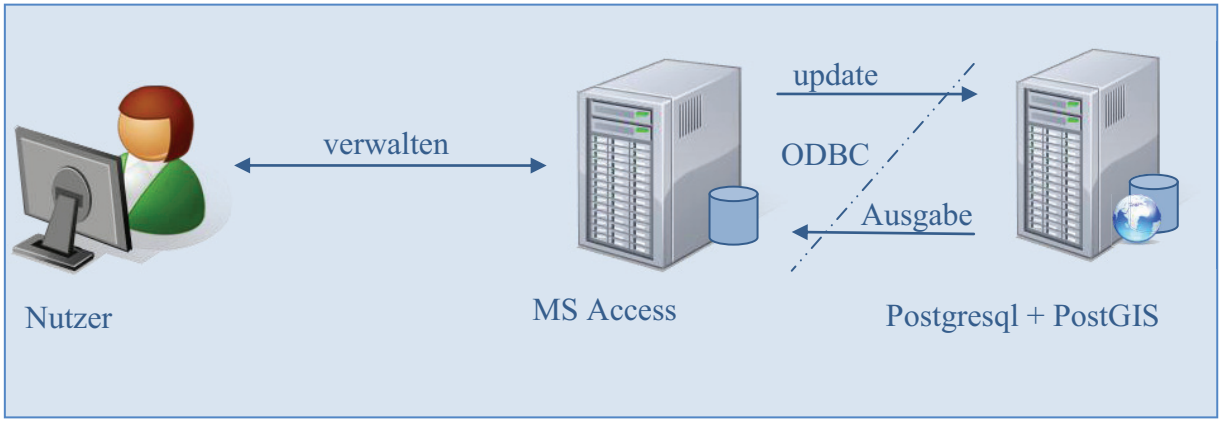


Abbildung 10 Verbindung zwischen Access und Postgresql über ODBC

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

8 Desktop GIS

Unter einem Desktop Geoinformationssystem (GIS) wird eine lokal zu installierende Software verstanden, mit deren Hilfe räumliche Informationen betrachtet, erzeugt, verarbeitet, gewonnen und verwaltet werden.

In der RFH GmbH befindet sich das proprietär ArcDesktop im Einsatz, welches durch die freie Software Quantum GIS abgelöst werden könnte.

8.1 ArcView

ArcView ist ein fester Bestandteil von ESRI´s GIS-Softwarepaket ArcDesktop. Mit ArcView wurde das de facto Standardformat Shapefile (siehe Abschnitt 9.1) für Geodaten eingeführt. In den neueren ArcView Versionen (9.x) wurde das Shapefile, durch die auf Microsofts Jet Engine basierende Personal Geodatabase(*.mdf), erweitert.

Die Geodatenverarbeitung lässt sich in ArcView durch einen hohen Funktionsumfang bewerkstelligen. Abgesehen vom Ändern, Löschen und Erzeugen von geometrischen Objekten wird die Datenpräsentation in 2D oder 3D Karten durch eine Menge an Vorlagen, Symbolen und einem Druckassistenten unterstützt. Zusätzlich besteht neben „einfachen“ Karten die Möglichkeit Daten, beispielsweise als Zeitreihen animiert, auszugeben.

Zur Datenverarbeitung und Präsentation lassen sich in ArcView nicht nur die hauseigenen Datenformate integrieren, sondern auch tabellarische Datenformate, CAD-Formate, Raster-Bilddatein, Webdienste, Multimediadatein und ca. 70 Industrieformate. Diese Daten können neben der Präsentationsmöglichkeit auch durch raumbezogene Analysen miteinander in Beziehungen gebracht werden. Um Zusammenhänge und Prozessmodelle zu erzeugen liefert ArcView eine Reihe von vorgefertigten Analysen und bietet die Möglichkeit durch selbst entwickelte Scripte Analysen an die eigenen Ansprüche anzupassen.

Einen weiteren Vorteil bringt der mitgelieferte Basisdatenkatolog von ESRI. In eigene Projekte können aus diesem Katalog thematische oder topographische Karten kostenlos eingebunden werden.

Sollte der Standardfunktionsumfang von ESRI nicht ausreichen, so lässt dieser sich durch kostenpflichtige Extensions erweitern.

8.2 OpenJump

OpenJump ist ein quelloffenes Desktop Geoinformationssystem und basiert auf einer Jumpentwicklung von Vivid Solutions. Der in Java geschriebene Quellcode ist auf allen Systemplattformen lauffähig und setzt lediglich Java in der Version 1.5 oder aktueller voraus.

OpenJump ist ein Vektor GIS, welches problemlos Rasterdaten einlesen und ausgeben kann. Geometriedaten können mit OpenJump verändert, gelöscht und erzeugt werden. Eine Analyse von Datensätzen ist sowohl nach geometrischen und logischen Parametern, als auch nach Sachattributen möglich. Als Datengrundlage können unter anderem GML, Shape und JPEG[44] Dateien dienen, aber auch die OGC Standards WMS und WFS werden in der aktuellen Version 1.3 unterstützt. Neben der Verwaltung von Diensten und Dateien besteht die Möglichkeit PostGIS Daten einzubinden, dabei bestehen die gleichen Bearbeitungsoperationen. Durch eine einfache Plugin-Struktur lassen sich mit entsprechenden Programmierkenntnissen zusätzliche Funktionen eigenständig hinzufügen, doch auch die Community bietet bereits eine Vielfalt an Zusatztools an.

8.3 QunatumGIS

Die Open Source Geospatial Foundation (OSGeo) hat mit Quantum GIS ein eigenes Desktop GIS Projekt ins Leben gerufen. Das in C++ geschriebene Programm ist bereits in der Version 1.1.0 kostenfrei im Quantum GIS-Portal[45] verfügbar. Eine Verwendung auf Uni, Linux Mac und Windows Computern wird durch den Einsatz der C++ Klassenbibliothek Qt-Toolkit ermöglicht.

Durch die Implementierung der OGR Bibliothek unterstützt Quantum GIS viele Vektorformate, wie z.B. ESRI´s Shapefile oder GML-Dateien. Rasterdaten, wie digitale Höhenmodelle oder JPEG Luftbilder, werden durch die eingebundene GDAL Bibliothek unterstützt. Das Einbinden von Spatiallight- und PostGIS-Datenbanken ist zusätzlich möglich und funktioniert ohne zusätzliche Systemkonfigurationen. Die eingebundenen Datensätze können editiert, gelöscht oder ergänzt werden. Die Analyse nach geometrischen und logischen Aspekten ist ebenso möglich, wie die Filterung nach bestimmten Sachdaten.

Ein wesentlicher Vorteil von Quantum GIS ist die benutzerfreundliche Speicherung von Projekten als Mapfile, so können diese unkompliziert online gestellt werden. Ein UMN Mapserver wird vorausgesetzt.

Zusätzlich besteht in Quantum GIS die Möglichkeit Projekte mit Zusatzelementen, wie Beschriftungen, Nordpfeil und Legenden zu versehen, um diese dann auszudrucken.

Erweitern lässt sich der Funktionsumfang von Quantum GIS durch in C++ oder Python geschriebene Plugins.

[44] Format für Bilder
[45] http://www.qgis.org

9 Visualisierung von Geodaten

Nachdem bereits Mapserver und Datenbanken beschrieben wurden, wird im folgenden Abschnitt die Shapedatei, der Web Map Service und der Web Feature Service vorgestellt. Diese drei Elemente dienen im Facility Management System der Visualisierung von Geodaten.

9.1 Shapefile

Das Shapefile entstammt der Softwareschmiede ESRI und wurde für die ArcView Produktreihe entwickelt. Dieses Format hat sich auf dem Geomarkt als de facto Standard durchgesetzt. Ein Shapefile ist immer ein Datenverbund von drei obligatorischen Dateien:

- *.shp speichert die Geometrien,
- *.shx verbindet Geometrien und Sachdaten über eine Indexstruktur,
- *.dbf im dBase-Format abgelegt Sachdaten.

Diese können durch die folgenden optionalen Dateien ergänzt werden:

- *.atx als Attributindex,
- *.sbx , *.sbn als Index für Tabellenverbindungen,
- *.aih und *.ain Index für Tabellenverknüpfungen,
- *shp.xml speichern die Metadaten vom Shapefile,
- *.prj Projektion der Daten,
- *.lyr Speicherung von Formatierungen (bis Version 3.3 *.avl).

Wie auch in der Layerdefinition des Mapfiles, kann eine Shapedatei nur eine Art von Geometrie verarbeiten. Zu den Geometriearten zählen auch hier Punkte, Linien oder Polygone.

9.2 Web Map Service – Ausgabe im Rasterformat

Wie im Abschnitt 3.4 erwähnt zählt, der Web Map Service (kurz WMS) zu den OGC Standards. Der WMS liefert georeferenzierte Bilder im Rasterdatenformat und Informationen zu diesen. Im Zusammenspiel mit einem Mapserver sind die folgenden drei verschiedenen Abfragen des Services möglich:

- GetCapabilties,
- GetMap und
- GetFeatureInfo.

Um einen WMS aufzurufen müssen die in Tabelle 2 genannten Parameter an den UMN Mapserver gesandt werden.

Parameter	Beschreibung
SERVICE	Angabe des Service, hier SERVICE=WMS
VERSION	Die Version des Service(1.0.0,1.1.0,1.1.1)
REQUEST	die im Text genannten WMS-Abfragen
REQUEST-Parameter	Parameter die von der Requestart abhängen

Tabelle 2 Pflichtparameter zur Einleitung eines WMS-Requests

9.2.1 GetCapabilities

Die GetCapabilities Schnittstelle wird von jedem OGC-Web Service unterstützt. Durch das Abrufen eines GetCapabilities-Requests erhält der Client alle Informationen über die Fähigkeiten und Eigenschaften des Dienstes. Als Resultat einer GetCapabilities-Abfrage wird dem Client ein XML-Dokument überliefert. Um dieses zu ermöglichen, müssen folgend die Abfrageparameter Service=WMS, Request=GetCapabilities und die Version=1.1.1(in Abhängigkeit der Verwendeten Version) festgelegt werden.

```
http://localhost/cgi-bin/mapserv.exe?map=mapfile.map&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetCapabilities
```

Beispiel 8 URL zum Aufruf eines GetCapabilities-Dokuments

In dem ausgelieferten XML Dokument können Metainformationen wie z.B. Name, Beschreibung, Ressource oder Kontakt abgerufen werden. Welche Metainformationen es gibt und welche unbedingt ausgefüllt sein sollten, sind in der ISO 19119 beschrieben.

Für einen gültigen WMS-Dienst, der über den UMN Mapserver abgerufen wird, werden die Metainformationen „wms_title“, „wms_onlineresource“ und „wms_srs“ im Mapfile hinterlegt. Der Geoserver hinterlegt standardisierte Default-Werte.

9.2.2 GetMap

Die GetMap Abfrage ist für die visualisierte Lieferung von Karteninformationen zuständig. Eine Ausgabe erfolgt in einem Bildformat, welches im Capabilities-Dokument enthalten ist. Um eine Karte vom WMS-Dienst zu erhalten ist in der Anfrage eine Reihe von obligatorischen (siehe Tabelle 3) und optionalen Parametern anzugegeben.

Parameter	Beschreibung
Dienst-Parameter	die grundlegenden Dienstparameter müssen angegeben werden (siehe Tabelle 2)
LAYERS	alle Layer die angezeigt werden sollen, Trennung durch Komma
STYLES	alle Styles die genutzt werden sollen, gleiche Reihenfolge wie die Layer
SRS	räumliches Referenzsystem z.B. SRS=epsg:2398

BBOX	BoundingBox-Koordinaten(links, unten, rechts, oben),gleiche Einheiten wie angegebenes SRS
WIDTH	Anzahl der Pixel in der Breite
HEIGHT	Anzahl der Pixel in der Höhe
FORMAT	Ausgabeformat des Kartenmaterials z.B. image/gif(unterstützt Transparenz) oder image/jpeg(keine Transparenz)

Tabelle 3 Pflichtparameter zur Ausführung eines GetMap-Requests

Durch die Überlagerung von mehreren WMS-Diensten können mehrschichtige Karten erstellt werden. Dazu müssen lediglich die Bounding Box, das Referenzsystem und die Kartengröße übereinstimmen. Damit eine optimale Anpassung gewährleistet ist, kann das Attribut *TRANSPARENT* für bestimmte Layer gesetzt werden. In dem Beispiel 9 wird als Hintergrundkarte eine Übersichtskarte des Geoportals Mecklenburg Vorpommern genutzt. Die Abfrage sieht dafür wie folgt aus:

```
http://www.gaia-mv.de/dienste/gdimv_dtk?&SERVICE=WMS&VERSION=1.1.0&REQUEST=GetMap&layers=gdimv_dtk&BBOX=4505385.080,5998305.500,4506040.900,5999781.060&SRS=epsg:2398&Format=image/gif&width=300&height=300&BGCOLOR=ffffff&STYLES=,,
```

Beispiel 9 URL zu einem GetMap-Aufruf eines systemfremden Dienstes

Und als Informationskarte, wird eine Karte vom eigenen System in Form eines WMS-Dienstes bereitgestellt. Die Abfrage, die die zweite Karte liefert,sieht dabei wie folgt aus:

```
http://localhost/cgi-bin/mapserv.exe?map=C:/ms4w/Apache/htdocs/rfh_mapfish/Anwendung/map/mapfileuebersicht.map&SERVICE=WMS&VERSION=1.1.0&REQUEST=GetMap&layers=Gebaeude&BBOX=4505385.080,5998305.500,4506040.900,5999781.060&SRS=epsg:2398&Format=image/gif&width=300&height=300&BGCOLOR=ffffff&TRANSPARENT=TRUE&STYLES=,,
```

Beispiel 10 URL zu einem GetMap-Aufruf eines Systemeigenen Dienstes

9.2.3 GetFeatureInfo

Die Abfrage GetFeatureInfo liefert auf Basis eines Koordinatenpaares die dazugehörigen Sachdaten. Zum Beispiel können Adresse und Bewohner eines Hauses ausgegeben werden, wenn auf einer Karte ein Gebäude angeklickt[46] wird. In Tabelle 4 sind alle Pflichtparameter für eine Anfrage aufgelistet.

[46] wenn von einem „Klick“ geschrieben wird, ist davon auszugehen dass die Nutzung der Maustasten gemeint ist

Parameter	Beschreibung
Dienst-Parameter	die grundlegenden Dienstparameter müssen angegeben werden (siehe Tabelle 2)
GetMap-Parameter	Alle Pflichtparameter des GetMap-Request müssen auch hier übergeben werden
QUERY_LAYERS	Angabe der Layer deren Informationen ausgegeben werden sollen
X	Pixel Koordinate des Bildes, gemessen von der linken Bildkante
Y	Pixel Koordinate des Bildes, gemessen von der oberen Bildkante

Tabelle 4 Pflichtelemente zur Ausführung eines GetFeatureInfor-Requests

Auch die Ausgabe von Sachinformationen erfolgt in den Formaten, die im Capabilities-Dokument hinterlegt worden sind.

```
http://localhost/cgi-bin/mapserv.exe?map=C:/ms4w/Apache/htdocs/rfh_mapfish/anwendung/map/mapfileuebersicht.map&SERVICE=WMS&VERSION=1.1.0&REQUEST=getFeatureInfo&layers=gebaeude&BBOX=4505324.21875,5998701.9140625,4506215.78125,5999078.0859375&SRS=epsg:2398&STYLES=&Query_Layers=Gebaeude&WIDTH=1268&HEIGHT=535&x=712&y=195&FEATURE_COUNT=30&INFO_FORMAT=text/html&FORMAT=image/png&STYLES=
```

Beispiel 11 URL zur Informationsausgabe bestimmter Objekte

9.3 Web Feature Service – Ausgabe im Vektorformat

Für die Ausgabe von Vektordatensätzen hat das OGC den Web Feature Service eingeführt. Diese Schnittstelle bietet neben dem Lesezugriff, auch das Erzeugen, Löschen, Aktualisieren und Sperren von geometrischen Objekten. Das WFS-Interface[47] ist komplett in XML gehalten worden. Vorteilhaft ist, dass bei der Bearbeitung nur die in den Metainformationen festgelegte Datenstruktur zu erkennen ist. Somit erfolgt das Speichern von Änderungen im Datensatz ohne das Wissen über die auf dem Server liegenden Datenstrukturen. Innerhalb des WFS können drei Arten des Dienstes unterschieden werden, bei denen die Anzahl der möglichen Operationen variiert.

- Basic WFS

[47] Interface ist eine Systemschnittstelle die dem Datenaustausch dient

- GetCapabilities
- DescribeFeatureTyp
- GetFeatureTyp

⇨ READ-ONLY-WFS

- XLink WFS
  - zusätzlich GetGmlObject

⇨ erweiterter READ-ONLY-WFS

- Transaction WFS
  - zusätzlich Transaction
  - LockFeature

⇨ READ&WRITE WFS

Um einen WFS abzurufen, müssen an den Mapserver die in Tabelle 5 aufgezeigten Parameter übergeben werden. Im Folgenden werden die Operationen des im Facility Management System angewendeten Basic WFS beschrieben.

Parameter	Beschreibung
SERVICE	Angabe des Service, hier SERVICE=WFS
VERSION	Die Version des Service(1.0.0,1.0.1)
REQUEST	die im Text genannten WFS-Abfragen
REQUEST-Parameter	Parameter die von der Requestart abhängen

Tabelle 5 Pflichtelemente die einen WFS-Request einleiten

9.3.1 GetCapabilities

Der GetCapabilities-Request muss für eine erfolgreiche Abfrage mit Hilfe der Methode Get-KVP integriert sein. Das WFS-GetCapabilities Dokument enthält die Bereiche:

- Service Identification
  - ⇨ WFS-Informationen
- Service Provider
  - ⇨ Auskünfte über den Datenanbieter
- Operation Metadata
  - ⇨ Liste alle ausführbaren Operationen mit ihren Einschränkungen und den daraus resultierenden möglichen Parametern.
- FeatureType-Liste
  - ⇨ Aufzählung der Featuretypen, die SRS und der Operationen die gemeinsam genutzt werden können

- ServesGMLObjectType-Liste
  - ⇨ Ausgabe aller nicht von gml:AbstractFeatureType abgeleiteten GML-Objekttypen, die mit der Operation GetGMLObject ausgegeben werden können.
- SupportsGMLObjectType-Liste
  - ⇨ Auflistung der GML-Objekttypen die WFS unterstützte GML-Schemata entsprechen
- FilterCapabilities
  - ⇨ Angabe alle Filteroperationen, Kombination von mehreren Filtern durch Logische Operatoren möglich

Diese Bereiche eines WFS-Capabilities-Dokumentes sind in der, vom OGC herausgegebenen, WFS-Spezifikation in Struktur und Bedeutung näher erklärt. Um einen WFS einzusetzen, muss der Client in der Lage sein Capabilities-Dokumente zu verarbeiten, um die Anfragen erfolgreich abarbeiten zu können.

9.3.2 DescribeFeatureType

Der DescribeFeatureType Request gibt Auskunft über die vom WFS angebotenen Features. Die Ausgabe enthält einen oder mehr Featuretypen, welche sich im Capabilities-Dokument befinden. Enthält die DescribeFeatureType Abfrage keine TypeName-Inhalte, so sind von jedem FeatureType Beschreibungen verfügbar. Das Schlüsselwort output-Format gibt an, in welchem Format die Beschreibungen ausgegeben werden. Standardisiert wird *text/xml;subtype=gml/3.1.1* ausgegeben, das heißt die Abfrage wird in GML3 zurückgeliefert. Im Capabilities-Dokument stehen auch alle MIME-Typen[48], die für das output-Format in Frage kommen.

9.3.3 GetFeature

Der getFeature-Request ermöglicht die Abfrage von sich auf Objekte beziehende Informationen. Diese werden dann unter Berücksichtigung vordefinierter und zusätzlich eingegebener Parameter ausgeliefert. Das Abfrageergebnis wird auch hier in Form eines XML-Dokumentes übergeben. Die visuelle Darstellung kann nur geschehen, wenn der Client XML-Dokumente interpretieren kann. Durch die Hilfe von Parametern werden die Abfragen eingeschränkt. Nach dem Einleiten des GetFeature-Requestes muss lediglich der typeName verpflichtend angegeben sein. Bei der Verwendung von Namen muss der Namespace verwendet werden (z.B. typeName=“rfh:gebaeude“).

[48] Multipurpose Internet Mail Extensions, beschreibt die Zusammensetzung eines Dokumentes

10 Mapfish

Mapfish ist ein WebGIS Framework des französisch-schweizerischen Unternehmens CamptoCamp. Das unter der GPL- und LGPL-Lizenz stehende Projekt basiert auf unterschiedlichen Open Source Produkten, die sich in serverseitige und clientseitige Programmparts aufteilen. Die Implementierung der Open Source Pakete unter Einbindung einer eigenen Programmbibliothek macht Mapfish zu einem WebGIS mit typischen GIS Funktionalitäten:

- Layer Manager,
- Toolbar,
- Suchfunktion,
- Editor,
- Geostatistik und
- Routing.

Aber auch GIS-fremde Funktionen können durch Mapfish in das WebGIS einfließen:

- Druckfunktionen und
- 3D Visualisierung.

Durch die Verbindung der unterschiedlichsten Open Source Produkte aus verschiedenen Kategorien (GIS, Design, Datenbank), ist eine zukunftsorientierte WebGIS-Lösung entstanden. Bei der Entwicklung wurde das Augenmerk auf die Variabilität von Mapfish gelegt. Dadurch lässt sich Mapfish mit Programmierkenntnissen an eigene Ansprüche sowohl optisch als auch funktional anpassen.

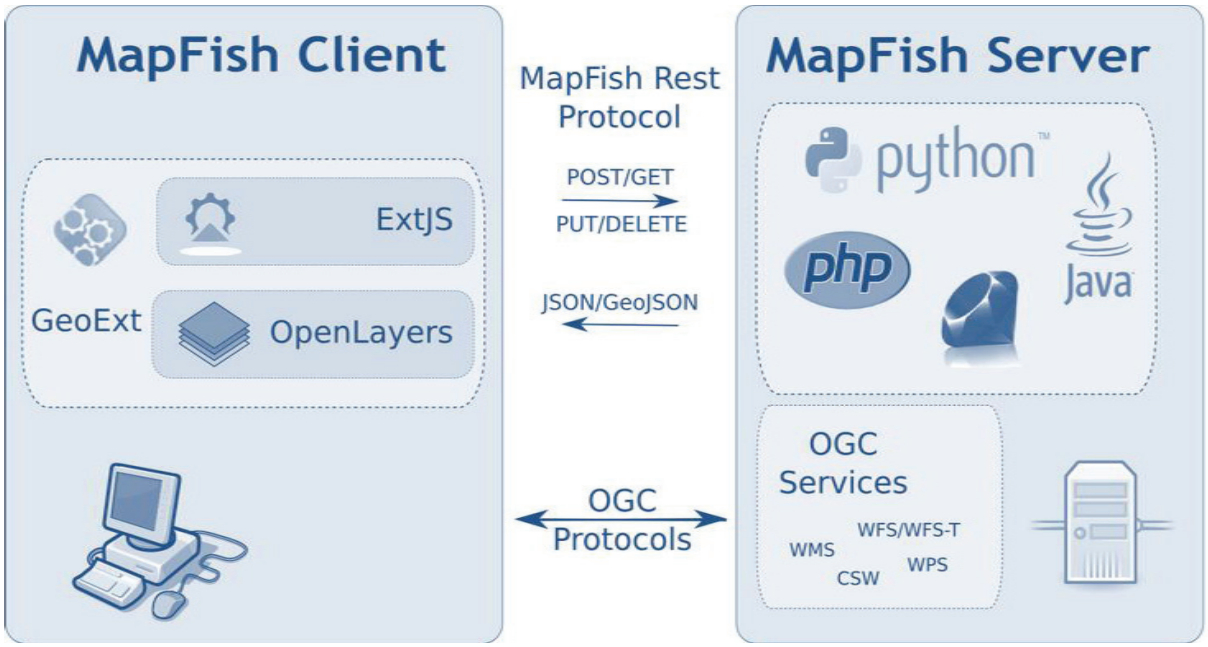


Abbildung 11 Struktur von Mapfish [5]

Software	Aufgabe
ExtJS	Bietet den optischen und funktionellen Rahmen für die Webanwendung(siehe 10.1.1)
OpenLayers	Dynamische Karteninhalte, die aus unterschiedlichen Quellen geladen werden können(siehe 10.1.2)
GeoExt	Vereint Karteninhalte mit ExtJS-Komponenten (siehe 10.1.3)
Shapely	Pythonpaket zur Analyse und Veränderung von 2D-Geometrien
SQLAlchemy	Einbindung der gesamten SQL Funktionalitäten
TileCache	Python basierter WMS-C und TMS-Server
PostGIS	Räumliche Datenbank im Backend
GeoJSON	Encodierung von vielen räumlichen Datenstrukturen
OGR	OGR Simple Features Library ist eine C + + Open-Source-Bibliothek stellt lesen und schreiben manchmal Zugriff auf eine Vielzahl von Vektor-Dateiformate
GDAC	Übersetzer-Bibliothek für Raster-Geodaten-Formaten
Pylons	Pylonen ist ein leichtes, Web-Framework betonen Flexibilität und schnelle Entwicklung

Tabelle 6 Bestandteile von Mapfish und ihre Aufgaben

10.1 Mapfish-Client

Der Mapfishclient ist ein Programmpaket, bestehend aus den 4 JavaScript-Programm-Bibliotheken von Ext JS, OpenLayers, GeoExt und Mapfish JS. Im Folgenden soll beschrieben werden, welche Funktionen diese innerhalb der Mapfishanwendungen einehmen.

10.1.1 Ext JS

Ext JS ist eine clientseitige API[49] zur Darstellung von rich-internet-applications[50]. Diese Webanwendungen ähneln wegen ihrem hohen Funktionsumfang und ihrem Windows Vista[51] ähnlichem Erscheinungsbild, eher einer Desktopanwendung als einer Webseite. Die verwendete Cross-Browser[52] JavaScript-Bibliothek ermöglicht eine nahezu identische Darstellung der Anwendungen in allen aktuellen Browsern. Basierend auf der Ereignis-Programmierung und der vollen AJAX-Unterstützung werden nur die DOM-Elemente

[49] steht für Aplication (deutsch: Anwendung)
[50] Anwendungen, die dem Nutzer umfassende Funktionen bieten in einer modernen Oberfläche anbieten. Verwendung von modernen Internettechniken, wie z.B. JavaScript und AJAX
[51] Betriebssysem von Microsoft
[52] CSS und JavaScript-Code wird in jedem Browsertyp nahezu identisch angezeigt

angezeigt, die aktuell in der Anwendung verwendet werden. Das Einbinden von externen Daten ist zu jeder Zeit möglich. Ein Browserereignis, wie zum Beispiel das Drücken eines submit-Buttons, führt zur Aktivierung einer JavaScriptfunktion, die zum Beispiel mit einem AJAX-Request dynamisch Inhalte nachladen kann und diese mit Hilfe von CSS-Vorlagen diese formatiert in die Webanwendung einbindet.

```
<script type="text/JavaScript">
    Ext.onReady(function() {
        new Ext.Viewport({
            layout: 'border',
            items: [
                {   region: 'north',
                    html: '<h1 class="x-panel-header">Page Title</h1>',
                    autoHeight: true,
                    border: false,
                    margins: '0 0 5 0'},
                {   region: 'west',
                    collapsible: true,
```

Beispiel 12 Ausschnitt der Hauptfunktion

Wie bereits angedeutet wird innerhalb der Ext JS API zwischen Optik und Funktionen unterschieden. Die JavaScript Bibliothek enthält alle Elemente zur Darstellung und zur Handhabung von Ereignissen, während in den CSS-Dateien die Elementformatierungen hinterlegt sind. An Elemente der Ext JS Umgebungen können auch Fremdfunktionen aus anderen JavaScript Anwendungen angebunden werden.

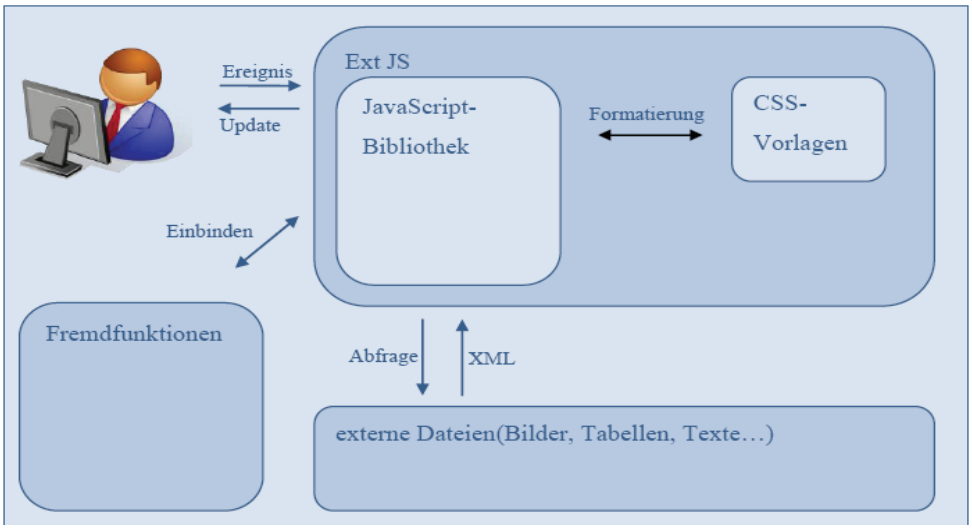


Abbildung 12 Funktionsweise von Ext JS

10.1.2 OpenLayers

Mit Hilfe von AJAX stellt OpenLayers Geodaten clientseitig im Browser dar. Entwickelt wurde OpenLayers anfänglich von MetaCarta. Die aktuelle Version 2.8 ist auf der OpenLayers-Homepage[53] für den eigenen Gebrauch zur Verfügung gestellt worden. Auch der offen gelegte Quelltext mit Beschreibungen von Funktionen und Parametern ist auf der Homepage einzusehen. OpenLayers besitzt zwar kein eigenes Kartenmaterial, ermöglicht aber das Einbinden von externen OGC konformen Webservices, wie WMS, WFS und WFS-T.

```
<script type="text/JavaScript" src="../Dateipfad/OpenLayers.js"> </script>
```

Beispiel 13 Integration der OpenLayers Bibliothek in die Anwendung

Um eine Kartenanwendung auf einer Webseite zu realisieren, reicht die Integration der OpenLayers Bibliothek (siehe Beispiel 13) und die Einbindung von wenig JavaScript-Code (siehe Beispiel 14). Innerhalb der Mapfish-Anwendung ist OpenLayers für die Einbindung und Darstellung von Geodaten zuständig. Viele Kartenoptionen, wie z.B. die SelectFeature Methode oder die Flächenmessung werden aus dem Pool der OpenLayersfunktionen übernommen und in die Mapfishapplication integriert.

```
JavaScript:
map = new OpenLayers.Map('map', {
                controls: [
                   new OpenLayers.Control.Navigation(),
                   new OpenLayers.Control.PanZoomBar(),
                ],
                numZoomLevels: 6
}
HTML:
<div id="map" class="map"></div>
```

Beispiel 14 Zusammenspiel zwischen HTML und JavaScript zur Kartendarstellung

10.1.3 GeoExt

Viele Ext JS-Module sind nicht kompatibel zu den OpenLayersfunktionen. Ab der Mapfishversion 1.2 versucht GeoExt die beiden JavaScript Bibliotheken von OpenLayers und Ext JS miteinander zu verbinden. Es werden Inhalte Sach- und Geodaten miteinander synchronisiert, ohne dass zusätzliche Funktionen geschrieben werden müssen. GeoExt dient

[53] http://www.openlayers.org

lediglich als Aufsatz und erweitert bestimmte Ext-Funktionen um räumliche Kompatibilitätsparameter. Auch umgekehrt werden OpenLayers Funktionen mit Ext JS Komponenten gekoppelt. Zum Beispiel kann eine Attributtabelle mit den Features eines Vektorlayers verbunden werden. Diese Abstimmung hat den Vorteil, dass mit einer Änderung der Tabelle oder des Vektordatensatzes die zweite Komponente synchron reagiert. In der aktuellen Version von GeoExt sind im Vergleich zu OpenLayers und Ext JS nicht viele Funktionen enthalten. In der Zielliste der GeoExt Entwickler sind aber hohe Ziele angepriesen, die den Einsatz von Mapfish noch deutlich erweitern sollen. Die Realisierung des GeoExt-Projektes treiben Entwickler von CamptoCamp-, OpenGeo und freiwilligen Programmieren voran. Eine aktuelle Version ist auf der Produkthomepage[54] kostenlos verfügbar, ebenso wie eine Beschreibung des Codes.

10.1.4 Mapfish JS

Die JavaScript Bibliothek übernimmt die Funktionen, die zu Beginn der Entwicklungsphase von keiner der bereits beschriebenen Bibliotheken übernommen werden konnten. Die meisten dieser Funktionen wurden mit der Entwicklung der GeoExt Bibliothek übernommen.
Zu den Funktionen die aktuell sowohl in GeoExt, als auch im Mapfish Code enthalten sind zählen:

- Recenter
- Legende
- Layertree

Zu den nur durch Mapfishcode implementierbaren Funktionen zählt das Drucken (setzt eine Erweiterung von Tomcat voraus) und die 3D Ansicht von Objekten.

10.2 Mapfish-Server

Der Mapfish-Server ist ein Framework auf dessen genaue Funktion nicht weiter eingegangen wird, weil er keine Verwendung im Facility Management System des RFHs findet. Anbei soll nur ein kurzer Überblick über seine Bestandteile und dessen Funktionen gegeben werden.

Bestandteil	Beschreibung
Shapely	Pythonpaket zur Analyse und Veränderung von 2D-Geometrien
SQLAlchemy	Einbindung der gesamten SQL Funktionalitäten
TileCache	Python basierter WMS-C und TMS-Server
PostGIS	Räumliche Datenbank im Backend

[54] http://www.geoext.org

GeoJSON	Encodierung von vielen räumlichen Datenstrukturen
OGR	OGR Simple Features Library ist eine C + + Open-Source-Bibliothek stellt lesen und schreiben manchmal Zugriff auf eine Vielzahl von Vektor-Dateiformate
GDAC	Übersetzer-Bibliothek für Raster-Geodaten-Formaten
Pylons	Pylonen ist ein leichtes, Web-Framework betonen Flexibilität und schnelle Entwicklung

Tabelle 7 Bestandteile des Mapfishservers

Umsetzung

11 Struktur des Facility Management Systems

In diesem Kapitel wird der genaue Aufbau des zukünftigen Facility Management Systems des RFHs vorgestellt. Das heißt es wird auf die zu installierenden Komponenten auf der Server- und der Clientseite eingegangen, sowie die Neuerungen des Arbeitsablaufes vorgestellt.

11.1 Serveraufbau

Auf der Serverseite befinden sich die Programme die im Hintergrund für die Lauffähigkeit der neuen programmtechnischen Verwaltungsstrukturen zuständig sind. Nach der Installation und Konfiguration dieser Programme sollte, ausgenommen bei Erweiterungen und grundlegenden Änderungen des Facility Management Systems, kein Bedarf an der Manipulation dieser Programme bestehen.

Das neue Herzstück der neuen RFH Strukturen ist die Postgresql-Datenbank. Diese dient zur Zusammenführung, Speicherung und Verwaltung der Sach- und Geodaten. Die Entscheidung für Postgresql fiel vor allem auf Grund von mangelnden Alternativen im Open Source Bereich, die sich auf einfache Weise mit Extensions um räumliche Funktionen erweitern lassen. Die räumliche Erweiterung für Postgresql bietet PostGIS und kann durch den Postgresql Stack Builder problemlos integriert werden.

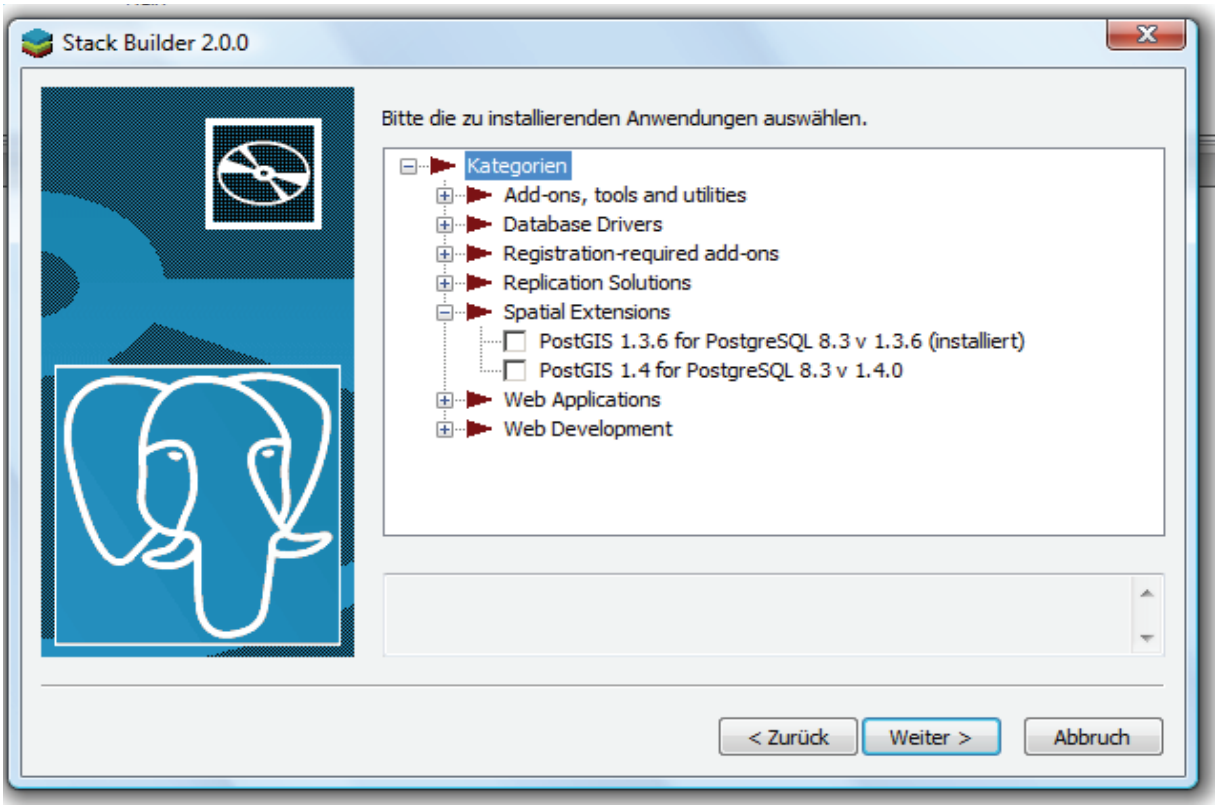


Abbildung 13 Stack Builder zur Erweiterung von Postgresql

Weitere Merkmale, die für eine Verwendung von Postgresql sprechen, sind die einfache Installation über einen Windows Installer und die große Community die über Foren Hilfe anbietet. Zur Einrichtung der Datenbankumgebung sind keine Programmierkenntnisse von Nöten und die Einrichtung kann über die mitgeliefert, graphische Benutzeroberfläche pgAdminIII durchgeführt werden. Dieses Tool kann auch für die weitere Bearbeitung benutzt werden. Durch die Verbindung mit Quantum GIS und der Microsoft Datenbank Access über einen ODBC-Treiber wird Postgresql lediglich im Hintergrund laufen und somit ein weiterer Kontakt nicht nötig sein. Die Verbindung zwischen Postgresql und MS Access muss im Vorfeld der Anwendung konfiguriert werden. Dazu muss unter *Systemsteuerung>Verwaltung>Datenquellen* eine neue Datenquelle angelegt werden. Das wird durch einen einfachen Klick auf *Hinzufügen* erreicht. Hier wird in der Liste relativ weit unten, *Postgresql35W Unicode* ausgewählt. Im daraufhin erscheinenden Menü werden die nötigen Parameter eingegeben und getestet. Nach bestandenem Test kann die Konfiguration mit save abgespeichert werden, die Verbindung zwischen Access und Postgresql ist nun möglich. Die Verbindung zwischen einzelnen Tabellen erfolgt im Datenbanksystem durch die Verwendung von externen Daten.

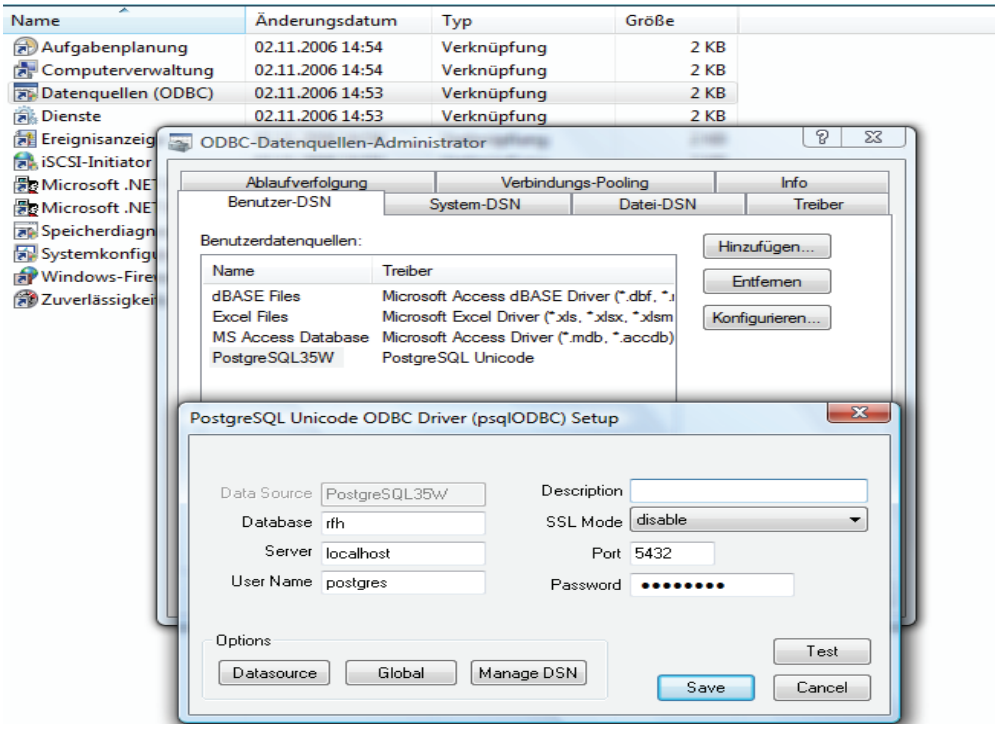


Abbildung 14 Einrichtung einer ODBC-Datenquelle

Als Webserver wird der http Webserver von Apache auf dem Server installiert. Die Gründe für die Wahl liegen bei den guten persönlichen Erfahrungen die mit diesem Webserver gemacht wurden. Die Konfiguration erfolgt über Konfigurationsdateien.

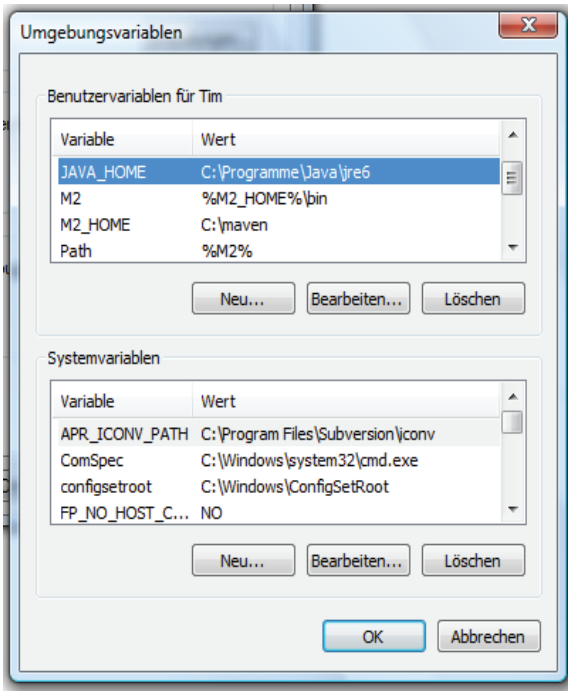


Abbildung 15 Umgebungsvariable für die Verwendung von Java

Damit auf der Serverseite javabasierte Programmcodes ausgeführt werden könnten, wird der Tomcatserver installiert. Um Tomcat nutzen zu können müssen die Umgebungsvariablen für Java festgelegt werden (siehe Abbildung 15).

Der Geoserver übernimmt die Aufgaben des Mapservers. Die Anwendung wird lediglich in das Webappsverzeichniss der Tomcats kopiert. Die weiteren Systemvorgänge übernimmt Tomcat, so dass keine aufwendige Installation nötig ist. Für den Geoserver und gegen den UMN Mapserver spricht die bereits integrierte, deutschsprachige und graphische Benutzeroberfläche mit deren Hilfe Geodaten auch ohne SQL-Programmierkenntnisse für die Präsentationsoberfläche zur Verfügung gestellt werden können.

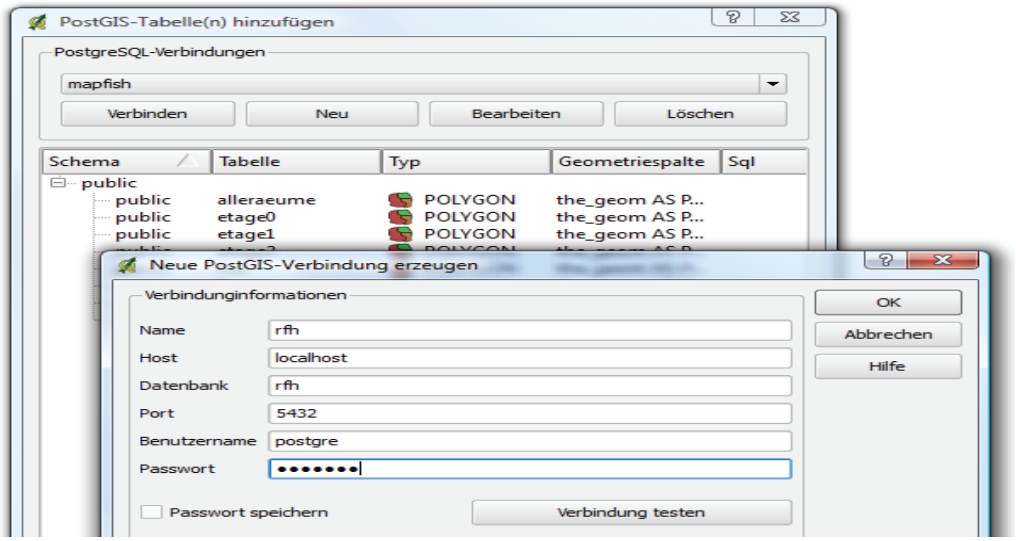


Abbildung 16 Herstellung einer Verbindung zwischen Postgresql und Quantum GIS

Die Datenpräsentation innerhalb des Facility Management System übernimmt Mapfish. Dieses zählt trotz der clientseitigen Ausführung zur serverseitigen Infrastruktur, weil die Dateien auf dem Server verwaltet werden und nur über den Webserver einem Client zur Verfügung gestellt werden. Die Installation auf dem Server wird durch einfaches kopieren der Mapfishdateien in das *htdocs* Verzeichnis des Webservers erzielt.

11.2 Clientaufbau

Auf der Clientseite sind die Programme installiert, die für die Verwaltung von Sach- und Geometriedaten sowie die Präsentation selbiger zuständig sind.

Clientseitig werden die Sach- und Geometriedaten der RFH Verwaltung eingegeben, gelöscht und aktualisiert. Für die Verwaltung der Sachdaten wird weiterhin die Oberfläche der MS Access Datenbank von Microsoft verwendet. Das bringt den Vorteil mit, dass die Sachdatenbearbeiterin Frau Paninka keine Einarbeitungszeit in eine neue Arbeitsumgebung benötigt, obwohl sie keine Daten mehr direkt in Access verwaltet. Denn die Daten werden über die Accessoberfläche in die, sich auf dem Server befindliche, Postgresql-Datenbank eingepflegt. Die Verbindung zwischen den beiden Datenbanken wird über den ODBC Treiber hergestellt, dessen Integration keine Probleme bereitet. Der weitere Einsatz von Access bedeutet, das es durch die Einführung der neuen Verwaltungsstrukturen keine Veränderungen der Arbeitsplätze bedarf, weil sich Access und Postgresql über das Intranet verständigen und nicht auf einem Arbeitsplatzrechner installiert sein müssen.

Für die Erzeugung und Aktualisierung der RFH Geometriedaten wird nicht mehr auf ESRI‘s ArcView zurückgegriffen, sondern, ohne den Verlust von Funktionen, Quantum GIS eingeführt. Für Quantum GIS und gegen ArcView spricht in erster Linie die unproblematische Verbindung mit Postgresql, die über einen Verbindungsassistenten hergestellt werden kann. Für den Einsatz von Quantum GIS sprechen zusätzlich die vielen, kostenlosen Erweiterungen, die unter anderem den Import und Export von Konstruktionsdaten im DXF-Format ermöglichen. Gegenüber ESRI Produkten, bietet die Forencommunity nur begrenzten und zeitlich deutlich länger anhaltenden Support[55], was aber nicht zwingend die Qualität mindert. Den Ausschlag zu Gunsten von Quantum GIS gegenüber OpenJump haben lediglich das modernere Layout und die Tatsache, dass sowohl die Webseite als auch das Produkt in regelmäßigeren Abständen aktualisiert werden, gemacht. Der Einsatz von Quantum GIS wird durch eine einfache Installation mit Hilfe eines Windows Installer ermöglicht.

[55] Hilfe oder Unterstützung

Als Browser und somit auch als Präsentationsplattform dient der Mozilla Firefox. Gegen den Internet Explorer von Microsoft hat sich Firefox auf Grund seiner Geschwindigkeitsvorteile im Entwicklungszeitraum durchsetzen können. Gegen Googles Browser Chrome spricht die undurchsichtige Datenschutzbestimmung in den AGBs, aus denen nicht eindeutig hervorgeht wann Daten an Google übermittelt werden. Dieser Nachteil konnte auch von den minimalen Geschwindigkeitsvorteilen Chroms gegenüber dem Firefox nicht aufgewogen werden. Ein weiterer Vorteil gegenüber seinen Konkurrenten hat der Firefox in seiner Vielfältigkeit, so kann er wie bereits beschrieben mit unzähligen Erweiterungen aufgewertet werden. Eine dieser Erweiterungen ist der Firebug, dessen Verwendung von der Mapfishcommunity geraten wird, weil sich die Hilfe oft auf Firebugexceptions bezieht. Die Installation dieses Browsers geht über eine knapp 4 MB große Installationsdatei, die lediglich ausgeführt werden muss.

11.3 Verwaltungsablauf

11.3.1 Sachdatenverwaltung

Die Sachdatenbearbeiterin Frau Paninka hat auf ihrem Arbeitsplatz das Datenbanksystem MS Access installiert und ein Intranetzugang. Die Datenbank des RFHs ist über die beiden Dateien Backend und Frontend zugänglich. Über die Frontend-Datei gelangt Frau Paninka in den Verwaltungsmodus mit einer unterstützenden Eingabemaske. Im Hauptmenü wird durch den Menüpunkt Datenbearbeitung (siehe Abbildung 17-Nummer 1), der Verwaltungsmodus aktiviert. Im zweiten Schritt wird die Wahl des zu verwaltenden Objekttyps festgelegt (siehe Abbildung 17-Nummer 2). In der daraufhin erscheinenden Maske besteht die Möglichkeit ein neues Objekt des gewählten Objekttyps zu erzeugen (siehe Abbildung 17-Nummer 3) oder über einen weiteren Klick die Gesamtheit alle gleichartigen Objekte einzusehen (siehe Abbildung 17-Nummer 4). Diese können innerhalb der tabellarischen Ansicht geändert oder gelöscht werden.

Durch die Verbindung der Datenbanksystem werden die Sachdaten der Objekttypen Flurstücke, Gebäude und Räume in der Postgresql gespeichert und nicht wie die restlichen Daten im Backend.

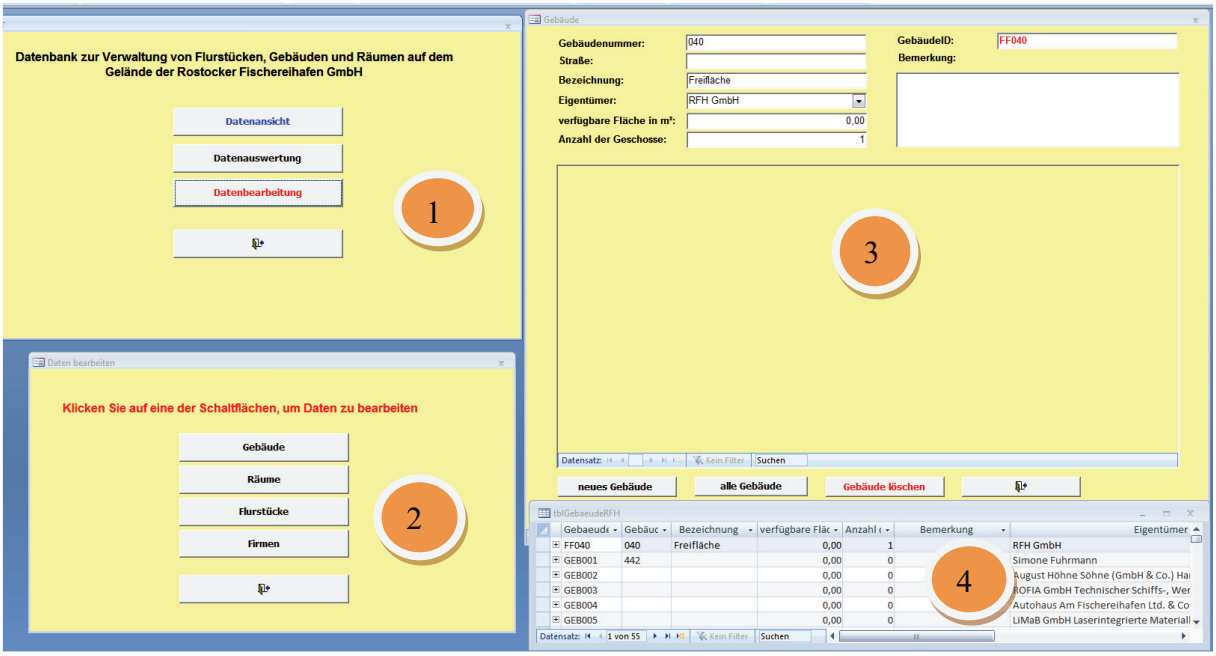


Abbildung 17 Screenshot der Eingabemaske von MS Access

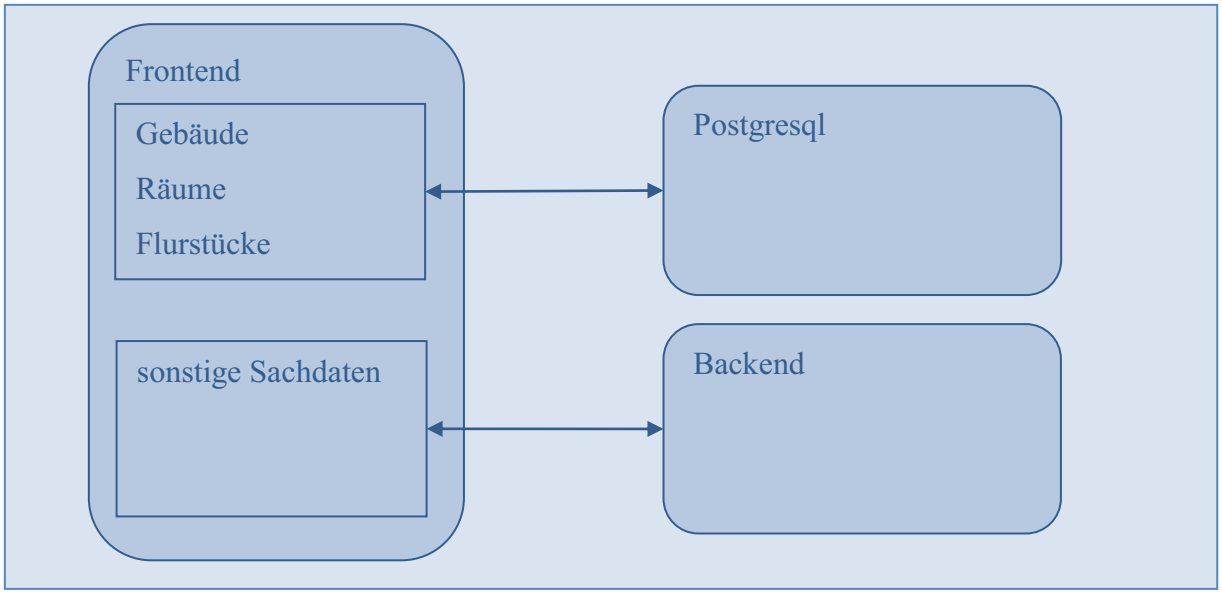


Abbildung 18 Verbindung zwischen Frontend, Backend von Access und Postgresql

11.3.2 Geodatenverwaltung

Die Verwaltung der Geodaten im RFH bleibt in den Händen von Frau Clausen, die aktuell diese Aufgabe, mit dem auf ihrem Arbeitsplatz installierten Desktop GIS QuantumGIS ausübt. QuantumGIS lässt sich wie die meisten Desktopanwendungen über ein Symbol auf dem Desktop oder in der Startleiste öffnen.

Nachdem Öffnen muss über die Schaltfläche *PostGIS-Layer*  eine Verbindung zur Objekte enthaltenden Datenbank aufgebaut werden. Dazu wird im erscheinenden Menü die Datenbank ausgewählt und über den *Verbinden*-Button die Verbindung hergestellt. Aus der

erscheinenden Liste wird die Tabelle geöffnet, deren Objekte aktualisiert, gelöscht oder hinzugefügt werden sollen (gebaeude für Gebäude, raeume für Räume und flurst für Flurstücke). Die Auswahl wird mit Hinzufügen bestätigt und erscheint im Hauptfenster von Quantum GIS.

Um der bestehenden Objektmenge ein Objekt zuzuführen, muss durch die Aktivierung der Schaltfläche *Bearbeitungsstatus ändern*  in den Bearbeitungsmodus gewechselt werden. Aus den daraufhin vom gräulichen ins farbige gewechselten Schaltflächen wird *Polygon digitalisieren*  ausgewählt. Durch diese Auswahl ist es möglich mit der linken Maustaste im Hauptfenster ein Polygon zu zeichnen, indem die gewünschten Eckpunkte gesetzt werden. Entspricht das Polygon nun den eigenen Vorstellungen bzw. den Vorgaben wird das Erstellen durch einen einfachen Rechtsklick abgeschlossen. Im daraufhin erscheinenden PopUp-Menü[56] können die Sachattribute für das generierte Objekt festgelegt werden, hier sollte lediglich der Objektschlüssel „Objektbezeichnung+id“ ausgefüllt und der Präsentationsstatus mit *true* oder *false* festgelegt werden. Um die Bearbeitung dauerhaft zu speichern, wird vom Bearbeitungsmodus zurück in den Informationsmodus gewechselt. Dies geschieht durch erneutes Betätigen der Schaltfläche *Bearbeitungsstatus ändern*  und anschließendes Wählen des im erscheinenden Kontextmenü Buttons *speichern*.

Um Objekte in Form und Größe zu ändern wird im Bearbeitungsmodus nicht auf Polygon digitalisieren sondern auf eine der folgenden Optionen geklickt:

- *Ring hinzufügen*  ,
- *Insel hinzufügen*  ,
- *Objekt verschieben*  ,
- *Objekt trennen*  ,
- *Stützpunkt verschieben*  ,
- *Stützpunkt hinzufügen*  oder
- *Stützpunkt löschen*  .

Auch diese Veränderungen werden erst in der Datenbank übernommen, wenn beim Umschalten des Bearbeitungsmodus gespeichert wird.

[56] ein Menü das durch eine Aktion in einem separaten Fenster auf dem Bildschirm erscheint

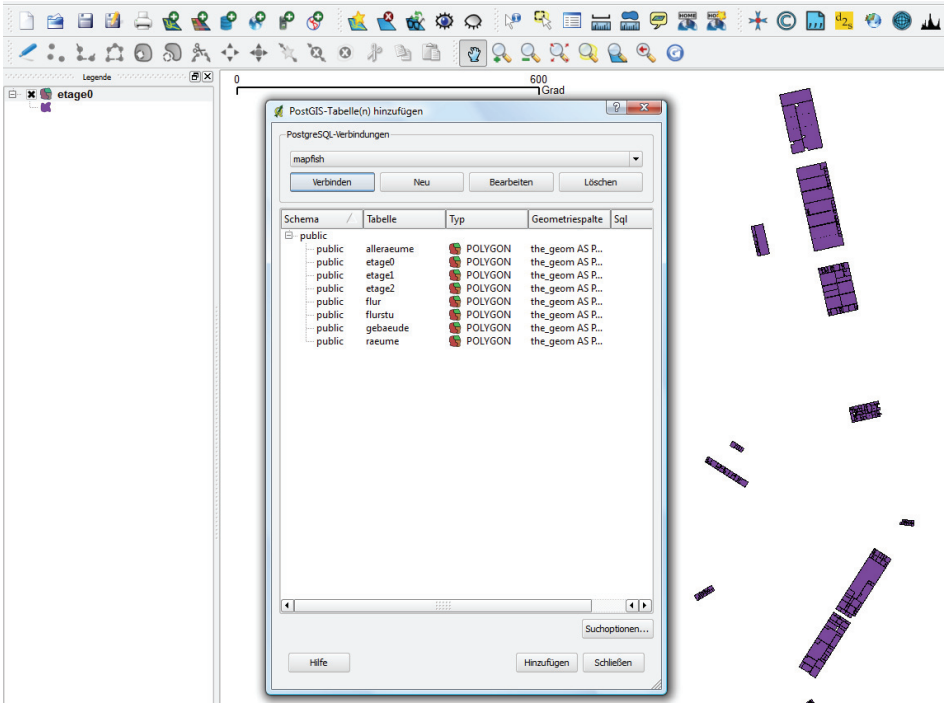


Abbildung 19 Einfügen von Postgres-Layern in Quantum GIS

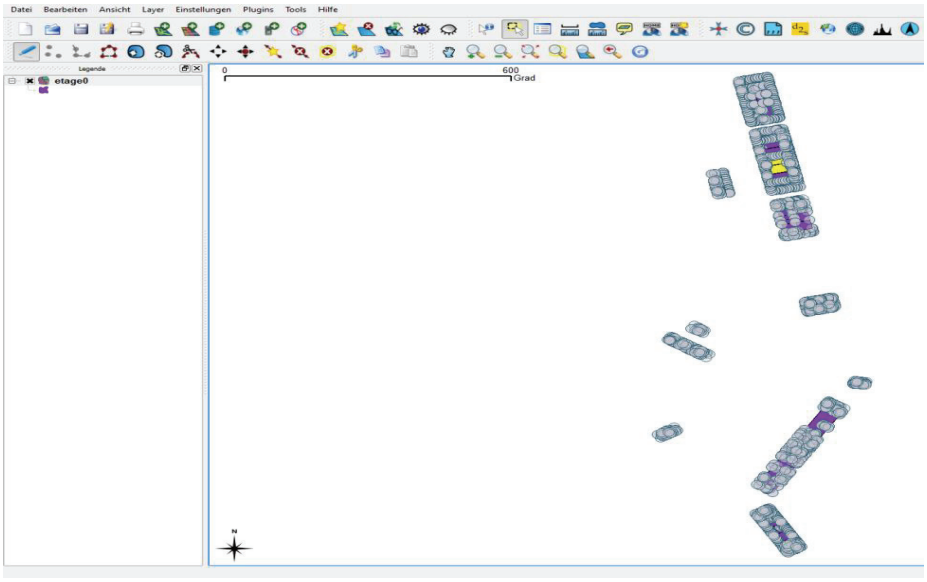


Abbildung 20 Selektionswerkzeug von Quantum GIS

Das Löschen einzelner Objekte wird im Bearbeitungsmodus erzielt, indem mit einem Selektionswerkzeug ein Objekt markiert wird und anschließend mit *Ausgewähltes löschen* entfernt wird. Wie gehabt wird diese Änderung beim Verlassen des Bearbeitungsmodus durch speichern übernommen.

11.4 Präsentationsablauf

Die Präsentationsplattform dient in erster Linie zur Darstellung der RFH-Geodaten und der Ausgabe der dazugehörigen Sachdaten unter Verwendung weiterer GIS typischer Funktion.

Um die Präsentationsplattform zu starten wird der Browser Mozilla Firefox gestartet und in der Adressleiste wird der Link aus dem Beispiel 15 eingetragen und mit Enter bestätigt.
Während der Nutzung bestehen folgende Nutzungsmöglichkeiten:

- Wahl eines Hintergrundbildes (Luftbild, Übersichtskarte, leere Karte),
- Anzeige von infrastrukturellen Objekten (Straßen, Beleuchtung, etc.),
- Anzeige von Räumen, Gebäuden und Flurstücken,
- Bemessung von Strecken und Flächen,
- Nutzung einer auf Filter basierten Suche,
- Ausgabe von Informationen zu einem Objekttyp in einer Liste,
- Drucken.

Auf die einzelnen Funktionen und deren Nutzung soll hier nicht näher eingegangen werden, dafür ist das Kapitel 12.3 vorgesehen.

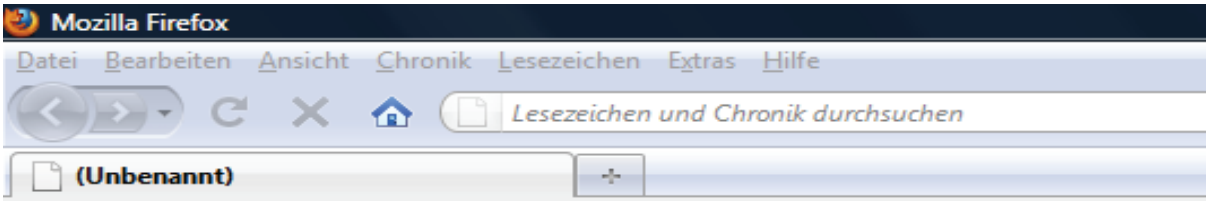


Abbildung 21 Adressfeld indem die Anwendung aufgerufen werden soll

```
http://Serverip/rfhmapfish/
```

Beispiel 15 URL zum Programmstart

12 Datengrundlage

In diesem Kapitel wird beschrieben wie, aus den vorhandenen Daten eine einheitliche Datengrundlage erschaffen wurde.

12.1 Datenbestand

Wie bereits erwähnt sind die Geo- und Sachdaten die für die Präsentationsplattform in Frage kommen, auf unterschiedlichen Arbeitsplatzrechnern gespeichert und weisen zudem unterschiedliche Dateiformate auf. Auf Grund dieser Tatsache sind die Daten vorerst vollkommen unabhängig voneinander zu betrachten.

Die Sachdaten der einzelnen Gebäude, Räume und Flurstücke sind in Tabellen in der Microsoft Access Datenbank hinterlegt. Die Datensätze sind dabei durch eindeutige Bezeichnungen als solche erkennbar. Jeder Datentupel ist durch einen eindeutigen Schlüssel gekennzeichnet und somit einem Objekt zuweisbar. Für die unterschiedlichen Objektklassen sind im Dateisystem verschiedene Sachdaten hinterlegt:

- Gebäude
  - GebaeudeID => eindeutiger Schlüssel
  - Gebaeudenummer
  - verfügbare Fläche in m²
  - Anzahl der Geschosse
  - Bemerkung
  - Eigentümer
  - Straße
- Räume
  - RaumID => eindeutiger Schlüssel
  - Gebäude
  - RaumNr
  - EtagenNr
  - Nutzung
  - NichtnuzbarM²
  - NebenM²
  - Größe in m²
  - Mietbeginn
  - Mietende
  - Mietvertrag

- Kaltmiete
- Betriebskosten
- Elektro
- Heizung
- Wasser
- Bemerkung
- Kaution

➢ Flurstücke

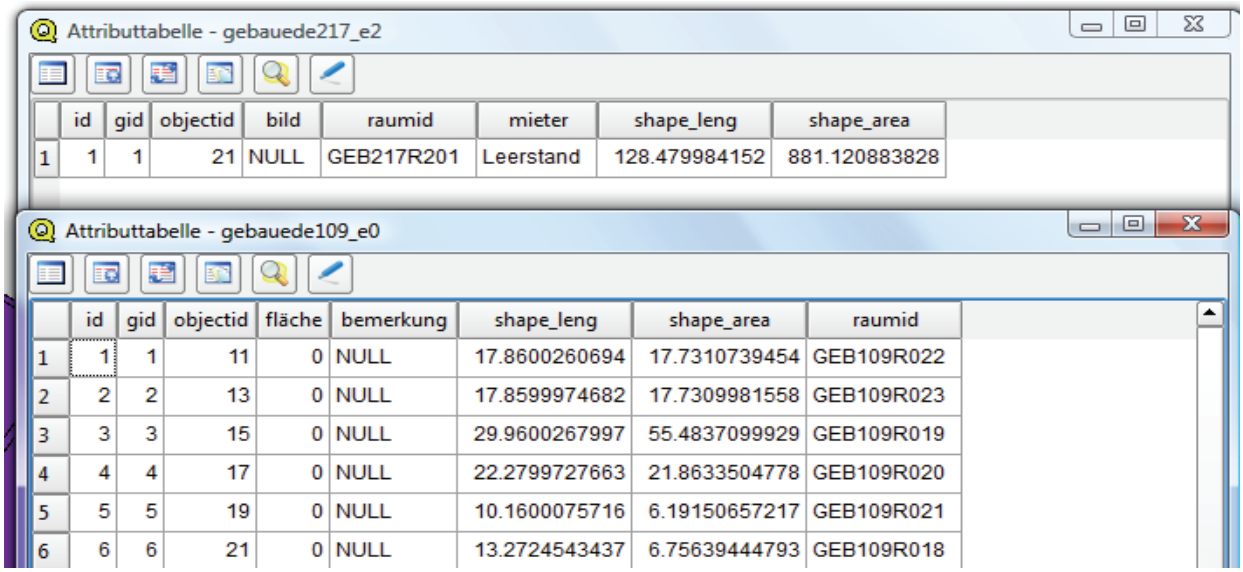
- Eigentümer
- lfdNrGrundbuch
- GMKG
- Flur
- Flurstück
- Fläche m²
- Fortführungsmitteilung vom
- Nutz lt Fortführungsmitt Kataster
- Lage
- Nutzung
- Stand
- LagehinweisRFH
- Bemerkung

Die Geodaten sind deutlich unaufgeräumter vorhanden, was jedoch damit zusammenhängt, dass diese Daten für verschiedene Zwecke mit unterschiedlichen Anforderungen erstellt worden sind.

Die Räume, Gebäude und Flurstücke, sowie zahlreiche weitere Objekte des RFHs sind in Shapedateien abgelegt. Neben den Geometriedaten sind in den Shapefiles auch Sachdaten an die Objekte angehängt worden, welche aber nicht zwingend mit denen der Accessdatenbank übereinstimmen.

Auffällig bei der Betrachtung der Shapefiles im QuantumGIS ist, dass viele Objekte nur als Hilfsobjekte dienen und somit für eine spätere Präsentation gar nicht in Frage kommen. Von den drei Objektklassen Gebäude, Räume und Flurstücke befinden sich lediglich die Flurstücke in einer eigenständigen Datei. Die Räume sind in Abhängigkeit ihrer Gebäude in separaten Shapes gespeichert und die Gebäude befinden sich in einer Nutzungsdatei des RFHs mit weiteren nutzungsrelevanten Objekten. Zwischen den Dateien für die Räume sind

unterschiedliche Attribute hinterlegt worden. Für alle Geodaten gilt, dass sie entweder im Koordinatensystem mit dem EPSG-Code 2398 oder ohne Koordinatensystem vorhanden sind.



Attributtabelle - gebauede217_e2

	id	gid	objectid	bild	raumid	mieter	shape_leng	shape_area
1	1	1	21	NULL	GEB217R201	Leerstand	128.479984152	881.120883828

Attributtabelle - gebauede109_e0

	id	gid	objectid	fläche	bemerkung	shape_leng	shape_area	raumid
1	1	1	11	0	NULL	17.8600260694	17.7310739454	GEB109R022
2	2	2	13	0	NULL	17.8599974682	17.7309981558	GEB109R023
3	3	3	15	0	NULL	29.9600267997	55.4837099929	GEB109R019
4	4	4	17	0	NULL	22.2799727663	21.8633504778	GEB109R020
5	5	5	19	0	NULL	10.1600075716	6.19150657217	GEB109R021
6	6	6	21	0	NULL	13.2724543437	6.75639444793	GEB109R018

Abbildung 22 Vergleich Sachattribute von verschiedenen Räumen

Weil neben den bereits genannten Objekten innerhalb der Präsentationsplattform die Möglichkeit bestehen soll, zu prüfen ob die Immobilien an die gegeben Infrastruktur angeschlossen sind, spielen auch die folgenden infrastrukturellen Objekte eine Rolle:

- Beleuchtung,
- Fernwärme,
- Regenwasser,
- Schmutzwasser,
- Trinkwasser,
- Straßen und
- Telefon.

Für jedes dieser Elemente gibt es eine Shapedatei mit den dazugehörigen Linien.

12.2 Datenerhebung

Um eine einheitliche Datenbasis zu schaffen müssen die bestehenden Sach- und Geodaten erst einmal in die Postgresql-Datenbank geladen werden, von wo aus dann eine Weiterbearbeitung erfolgt.

Für das Einspielen von Geodaten in eine Postgresql-Datenbank können neben der Eingabe per Hand, Quantum GIS und die shptopsql-Bibliothek verwendet werden. Die shptopsql-Bibliothek wird über die DOS-Konsole verwendet und legt

SQL-Insert-Statements[57] im Textformat ab. Diese Textdateien können nun über den Postgresql SQL-Reader[58] eingelesen werden.

QuantumGIS bietet eine deutlich komfortablere Lösung für das Exportieren von Shapedateien in eine Postgresql-Datenbank. Auf diese Methode der Datenüberspielung soll daher näher eingegangen werden. Dazu wird Quantum GIS auf herkömmliche Art und Weise geöffnet und der Exportassistent über die Schaltfläche mit dem blauen Elefanten  geöffnet. In dem sich öffnenden Menü wird die Datenbank ausgewählt, welche die Geodaten aufnehmen wird. Mit Hilfe der, mit *Hinzufügen* beschrifteten, Schaltfläche, im unteren Menübereich, wird die zu exportierende Shapedatei ausgewählt. Nachdem das *SRID* eingegeben wurde, kann das Exportieren mit *OK* gestartet werden.

Die Geodaten finden sich in einer eigenen Tabelle, mit dem Namen der Shapedatei, in der Postgresql-Datenbank wieder.

Die Sachdaten werden über das Backend der MS Access Datenbank exportiert. Dieses wird erreicht in dem in der Tabellenübersicht die gewünschte Tabelle mit der rechten Maustaste ausgewählt wird. Im erscheinenden Menü wird Exportieren und im Untermenü ODBC-Datenbank aktiviert. Daraufhin wird ein Tabellenname für die Postgresql-Datenbank eingegeben und bestätigt. Im folgenden Schritt wird im erscheinenden Fenster der Reiter[59] *Computerdatenquelle* aktiviert und daraufhin die vorkonfigurierte Postgresql Datenquelle ausgewählt. Mit OK wird das Exportieren bestätigt. Als Zieldatenbank wird die in der konfigurierten Datenbankverbindung angegebenen Postgresql Datenbank genutzt. In dieser ist die Tabelle mit den Sachdaten unter dem eingegebenen Namen zu finden.

12.3 Datenhomogenesierung

Im Anschluss an den Datenexport der Sach- und Geometriedaten, in die Postgresql-Datenbank, befinden sich diese in unterschiedlichen Tabellen wieder. Ziel ist es für die Räume, Gebäude und Flurstücke jeweils eine Tabelle mit Sach- und Geometriedaten zu erzeugen.

Um mit den Tabellen der Sachdaten arbeiten zu können müssen diese an die Bedingungen von Postgresql angepasst werden, das heißt die Spaltennamen sollten klein geschrieben sein und ohne Leer- und Sonderzeichen auskommen. Um das zu erreichen wird in der pgAdminIII-Oberfläche ein Tabellenname mit der rechten Maustaste angewählt und in dem erscheinenden Menü die Option Eigenschaften ausgewählt. Hier können die Spaltennamen

[57] SQL-Befehl der einen Datenimport ausführt
[58] Programmteil der pgIIIAdmin-Oberfläche der den SQL-Code liest und interpretiert
[59] dienen der Auswahl von überlagerten Dokumenten oder Menüs (ähnlich der Auswahl von Karteikarten)

über den Reiter *Spalten* durch einfaches Auswählen und zusätzliches Aktivieren der, mit *ändern* beschrifteten, Schaltfläche geändert werden.

Das Erstellen einer einheitlichen Tabelle ist für die Flurstücke am einfachsten zu realisieren. Dazu wird der Tabelle mit den Sachdaten der Flurstücke eine Spalte mit dem Datentyp *geometry* hinzugefügt. Dieses wird unter den bereits erwähnten Tabelleneigenschaften, ebenfalls im Reiter *Spalten* vollzogen, indem durch *Hinzu* eine weitere Spalte an die bereits bestehende Tabelle gehängt wird. Als Spaltenname wird *the_geom* gewählt und als Datentyp

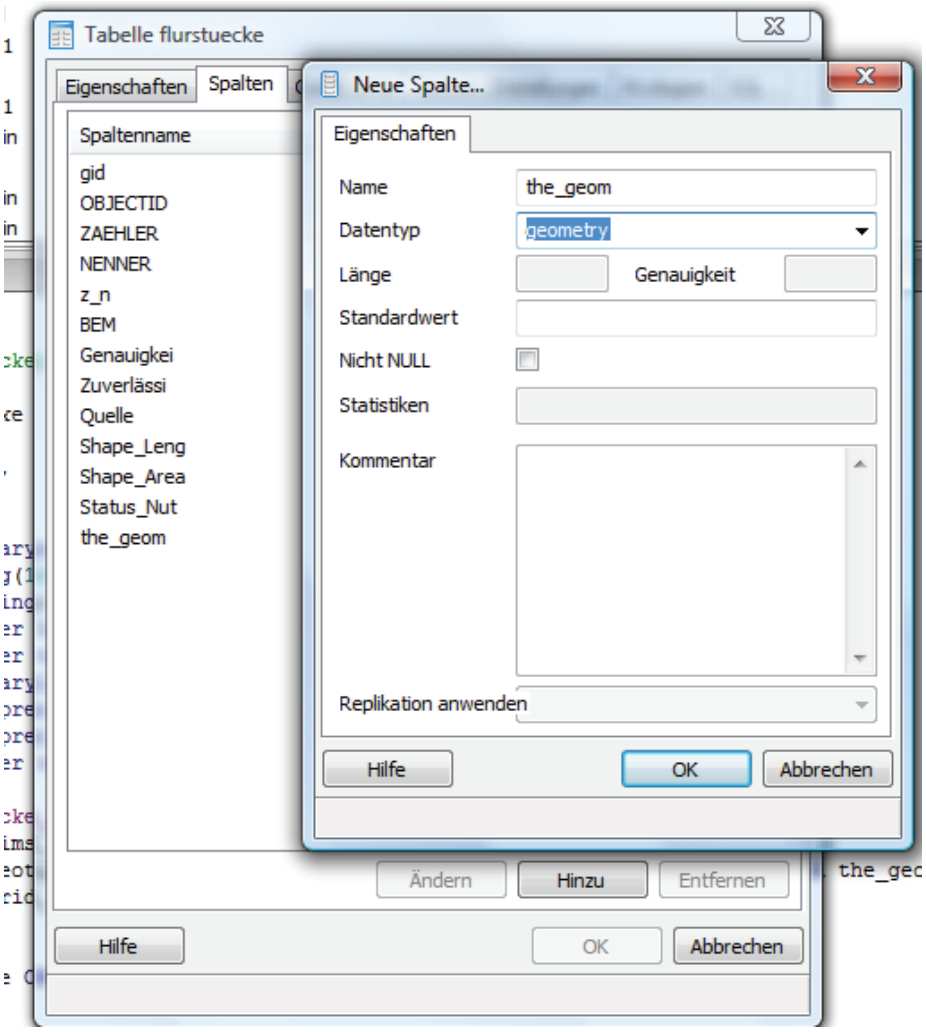


Abbildung 23 Festlegen einer Geometriespalte in Postgresql

```
update flurstueckesachdaten  as a set the_geom= b.the_geom from flurstuecke as b where a.flurstueckid=b.flurstueckid and  a.the_geom is null;
```

Beispiel 16 Einfügen der Geometriedaten in die Sachdatentabelle für die Flurstücke

geometry festgelegt. Im nächsten Schritt werden die Sachdaten durch die jeweiligen Geometrien ergänzt. Dazu werden die Geometriedaten aus der Tabelle des Shapefiles genommen und an die Stelle der Sachdatentabelle eingefügt an der die Primärschlüssel

identisch sind (siehe Beispiel 16). Dadurch ist eine einheitliche Tabelle mit den Attributwerten für die Flurstücke entstanden.

Auf ähnliche Art und Weise funktioniert die Erzeugung einer einheitlichen Tabelle für die Gebäude. Die Sachdatentabelle der Gebäude wird um die Geometriespalte erweitert, um dann die dazugehörigen Geometrien aus der Geometrietabelle einzufügen. Das SQL-Statement wird dabei nur um eine Komponente erweitert, denn die Geometrien der Gebäude kommen aus einer Tabelle mit mehreren Objektkategorien.

```
update gebaeudesachdaten  as a set the_geom= b.the_geom from gebaeude as b where a.gebaeudeid=b.gebaeudeid and  a.the_geom is null and b.art='gebaeude';
```

Beispiel 17 Einfügen der Geometriedaten in die Sachdatentabelle für die Gebäude

Die Homogenisierung der Sach- und Geometriedaten des Objekttyps Räume ist unwesentlich komplizierter. Der Unterschied zu den vorhergehenden Methoden ist das die Geometriedaten über mehrere Tabellen aufgeteilt sind. Es kann entschieden werden ob erst alle Geometrietabellen zusammengefasst werden oder das auch für die Gebäude und Flurstücke verwendete SQL-Statement für jede Tabelle einzeln aufgerufen wird. Aus Gründen der

```
create table zwischenschritt as
select raumid, the_geom from gebaeude109_e0 union
select raumid, the_geom from gebaeude109_e1 union
alle weiteren Geometrietabellen mit Räumen;

update gebaeudesachdaten  as a set the_geom= b.the_geom from zwischenschritt as b where a.raumid=b.raumid and  a.the_geom is null;
```

Beispiel 18 Einfügen der Geometriedaten in die Sachdatentabelle für die Räume

Übersichtlichkeit wird dazu geraten zu erst alle Raumgeometrien und deren Schlüsselattribute zusammenzuführen und anschließen die Sachdatentabelle mit den Geometriedaten zu ergänzen.

Bei allen drei Objekttypen gilt es zu beachten, dass die Schlüsselwerte der einzelnen Daten übereinstimmen. Jenes war in der Testphase nicht gegeben, wodurch viele Sachdaten ohne Geometrien übrig blieben. Dieses wird bis zum endgültigen Einsatz noch durch die beiden, in der Verwaltung tätigen, Damen Frau Paninka und Frau Clausen behoben.

12.4 Datenänderungen

Um das RFH Facility Management System zu verbessern, werden die vereinheitlichten Tabellen der Gebäude, der Räume und der Flurstücke um zwei Spalten erweitert.

Die erste zusätzliche Spalte wird mit *praesentation* bezeichnet und ist vom Datentyp *boolean*. Durch diese Spalte können die Tabellen auch weiterhin durch Hilfsobjekte ergänzt werden, ohne die homogenisierte Struktur zu verändern. Da diese weder in der MS Access Datenbank noch in der Präsentation auftauchen müssen. Zur Realisierung wird bei den Hilfsobjekten die Spalte *praesentation* auf *false* gestellt. Das Löschen dieser Elemente ist dadurch ebenfalls deutlich einfacher, denn im Quantum GIS können die Hilfsobjekte durch eine einfache Filterfunktion alleine angezeigt werden und somit gleichzeitig markiert und gelöscht werden. Dadurch wird vermieden, dass jedes Objekt einzeln angeklickt und kontrolliert werden muss. Die zweite ergänzende Spalte trägt den Namen *bild* und beinhaltet den Namen der Objektfotos. Diese Erweiterung dient zur Ergänzung der Objektinformationen um eine visuelle Information in der Präsentationsplattform auszugeben. Die Bilder werden standardmäßig in einem festgelegten Ordner auf dem Server gespeichert, wodurch die Angabe des Namens ohne Pfad erfolgt.

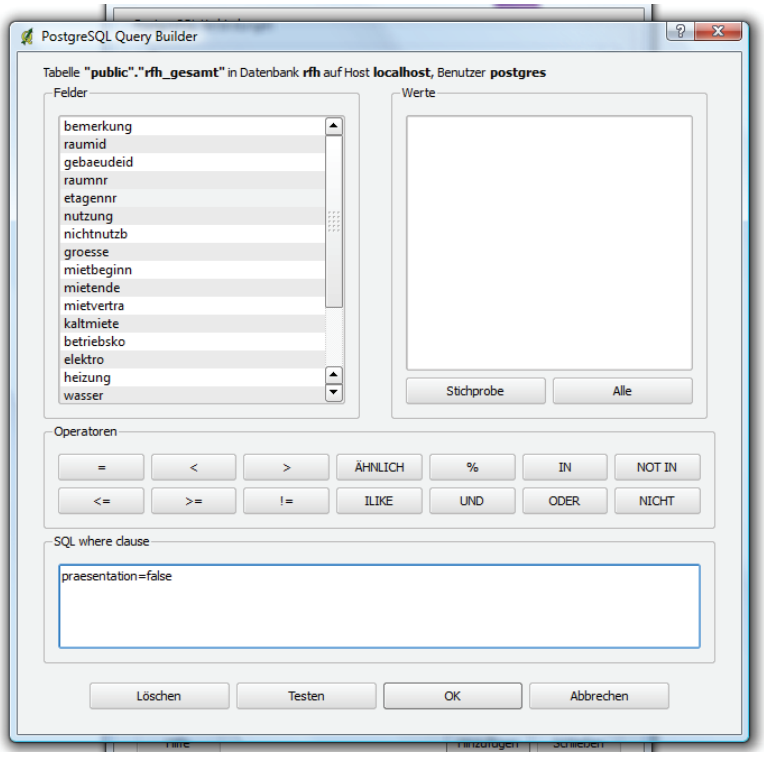


Abbildung 24 Einschränkung der aus Postgresql geholten Daten

12.5 Datenbereitstellung

Um die Eingabe der Daten zu vereinfachen und doppelte Eingaben zu vermeiden, wird über sogenannte *Views* auf die Basisdaten zugegriffen. *Views* sind dabei Sichten auf die Ausgangsdaten, die die Ansicht limitieren können und dem Nutzer durch bestimmte Regeln

Rechte einräumen. Durch die Nutzung von *Views* kann die Eingabe strikter nach der Sachdatenverwaltung und der Geodatenverwaltung getrennt werden.

Die Aufteilung erfolgt in einen Sachdatenview und einen Geodatenview für die einzelnen Bereiche der Verwaltung. Durch den Sachdatenview wird der Zugang zu den Basisdaten auf die Sachattribute beschränkt, das heißt es werden alle Attribute bis auf die Geometrie und die Werte der Präsentationspalte angezeigt. Der Sachdatenview erhält dabei die Rechte, die zu betrachtenden Werte zu ändern, zu löschen und zu ergänzen.

Ähnlich ist die Funktionsweise bei den Geodaten nur das hier bis auf die Geometrie, der Objectid und der Präsentationsspalte keine weiteren Werte ausgegeben werden. Auch für die Geodatenverwaltung stehen die Rechte des Löschens, des Änderns und zum Erzeugen zur Verfügung.

13 Mapfish

In diesem Kapitel werden die strukturellen Eigenschaften und die Funktionalitäten der neuen Präsentationsplattform des RFHs vorgestellt und erläutert.

13.1 Datenstruktur

Das Mapfish serverseitig einer einfachen Ordnerstruktur unterliegt und ohne aufwendige Installationsprozesse auskommt wurde bereits gesagt. Der Mapfishordner muss also lediglich in den Webordner des Webservers hineinkopiert werden.

Dieser Ordner enthält vier Unterordner und zwei separate Dateien. Der erste Ordner trägt den Namen *client* und beinhaltet alle Programmbibliotheken der Mapfish-API und zusätzlich die proj4js-Bibliothek auf die später näher eingegangen wird. Der mit *icons* bezeichnete Ordner enthält die Imagevorlagen für die Schaltflächen der Präsentationsplattform. Unter *images* werden die Bilddateien für die einzelnen Gebäude, Flurstücke und Räume abgespeichert. Im Verzeichnis *jsfunktionen* befinden sich die selbstgeschriebenen JavaScriptfunktionen, die teilweise auch auf Dateien des Clientordners zurückgreifen.

Bei den beiden sich im Hauptordner befindlichen separaten Dateien handelt es sich um die *style.css* Datei und die index.html Datei. Die *style.css* Datei enthält die Layoutinformationen für Anwendungselemente, die nicht dem Standarddesign unterliegen. Mit der *index.html* Datei wird die gesamte Präsentationsanwendung gestartet. Durch diese Datei werden alle nötigen Funktionen und Elemente zusammengeführt und angeordnet ausgegeben.

13.2 grundlegende Implementierungen

In diesem Abschnitt werden Funktionen beschrieben, die für die Anwendung von extentieller Bedeutung sind, aber während der Nutzung nicht direkt verwendet werden.

13.2.1 Mapobjekt

Das Mapobjekt ist das Hauptobjekt der Webanwendung. In ihm werden die Eigenschaften definiert, die die Rahmenbedingungen für die Benutzung der Webmapanwendung festlegen. Im Mapobjekt wird die Projektion angegeben, in der alle geometrischen Objekte angezeigt werden und die die Grundlage zum Beispiel für die Messungsfunktionen bildet. Weitere zu anzugebende parameter sind, unter anderem die Definition der möglichen Zoomskalen und die maximal angezeigte Kartenausdehnung des gesamten Projektes.

Das Mapobjekt wird mit seinen Eigenschaften über eine OpenLayersfunktion erzeugt.

```
Mapobjekt:
map = new OpenLayers.Map("map",{projection: "EPSG:2398",controls:[],scales:[
3460498.76953125…],maxExtent:new OpenLayers.Bounds
(4504884.5,5997999.5,4506172.5,5999757.5)});
```

Beispiel 19 JavaScriptcode für die Erzeugen des Mapobjektes

Über einfache OpenLayersfunktionen können dem Hauptelement Layer und Funktionen zugeführt werden.

```
Layer:
map.addLayer(Layername);
Bedienelemente:
mad.addController(Controllername);
```

Beispiel 20 Hinzufügen von Layern und Steuerungselementen

13.2.2 WMS Layer

Die WMS-Layer dienen in der RFH-Anwendung als Hintergrundbilder und zur visualisierten Ausgabe der lokal vorhandenen Infrastruktur. Die Implementierung der einzelnen Layer erfolgt über JavaScriptfunktionen, welche sich in der Datei *wms_layer.js* im jsfunktionen-Verzeichnis befinden. Innerhalb der einzelnen WMS-Funktionen werden Variablen erzeugt. Aus der Variablen selbst wird ein WMS-Layer, indem ihr die Openlayers.Layer.WMS Funktion übergeben wird. Der Funktion selbst müssen einige Parameter beigefügt werden. Das ist der Name des zukünftigen Layers, der WMS-Dienstherkunftslink. Der daraus zu verwendete Layername und weitere optionale Parameter. Bei einem der verwendeten Layer muss die Eigenschaft *isBaseLayer* auf *true* gesetzt werden, weil die OpenLayers-Bibliothek einen Basislayer voraussetzt. Um einen weißen Hintergrund zu erzeugen muss der WMS-Dienstlink leer gelassen werden und die Basislayereigenschaft auf *false* gesetzt werden.

Mit der bereits erwähnten Funktion *addLayer* wird der Layer dem Mapobjekt beigefügt. Die gesamte Hauptfunktion wird nun durch die *index.html* aufgerufen.

```
function createUebersichtLayer(map) {  //Funktionsname
uebersicht =  new OpenLayers.Layer.WMS( // Variable mit Funktion
            "uebersicht", //zukünftiger Layername
            "http://www.geodaten-mv.de/dienste/gdimv_dtk", //WMS Herkunft
            {layers: "gdimv_dtk"}, //Layername
            {isBaseLayer: true}// Basislayereigenschaft
        );
        map.addLayer(uebersicht);}
```

Beispiel 21 Erzeugen eines WMS Layers

13.2.3 WFS Layer

Die WFS-Layer dienen der Ausgabe der Räume, der Gebäude und der Flurstücke des RFHs im Vektorformat. Ähnlich wie bei den WMS-Layern werden die WFS-Layer durch separate Funktionen, die sich inder Datei *WFS_Layer.js* befinden, erzeugt.

Innerhalb dieser Funktionen werden den generierten Variablen die Funktion *OpenLayers.Layer.Vector* übergeben. Anders als üblich erhalten die Vektorlayer des RFHs in diesem Schritt lediglich ihre Darstellungsparameter, aber keine anzuzeigenden Objekte. Die Objektzuweisung an leere Layer wird in Kapitel 12.3.4 näher beschrieben. Die leeren Layer werden dennoch über die bekannte Funktion *addLayer* dem Mapobjekt übergeben. Innerhalb der *index.html* kann diese Funktion nun abgerufen werden.

```
function gebaeudestore(map){ //Funktionsname
gebaeude = new OpenLayers.Layer.Vector("gebaeude",{styleMap: gebaeudeStyle});
map.addLayer(gebaeude);}
```

Beispiel 22 Funktion zur Erzeugung eines WFS Layers

13.3 Funktionen

Im Folgenden werden die Funktionen der Präsentation in ihrem Nutzen, in ihrer programmiertechnischen Umsetzung und in ihrer Anwendung beschrieben.

13.3.1 Grundnavigation

Innerhalb des Kartenbereichs dient die Grundnavigation als schaltflächenlose Navigation, um nicht einsehbare Bereiche sichtbar zu machen. Der Kartenausschnitt kann durch einfaches ziehen mit der Maus hin und her bewegt werden und mit dem Drehen des Mausrades wird dieser vergrößert bzw. verkleinert.

Das Verschieben des Kartenbereichs funktioniert, durch das Bewegen der Maus inklusive gedrückter, linker Maustaste. Wird die Maus auf dem Bildschirm nach unten bewegt,wandert auch der Kartenausschnitt nach unten, wodurch der zusehende Kartenbereich nach Norden rückt.

Eine programmtechnische Implementierung dieser Funktion ist nicht notwendig, weil diese standardmäßig von Mapfish eingebunden ist.

Das Zoomen mit dem Mausrad funktioniert durch die Betätigung des Mausrades innerhalb des Kartenbereiches. Beim Nachvornedrehen wird der Ausschnitt vergrößter beim Zurückdrehen dementsprechend verkleinert.

Für das Nutzen der Scrollfunktion zum Zoomen ist die Implementierung einer Funktion notwendig. Die Funktion befindet sich in der Datei *mapoptions.js* und wird in der *index.html* aufgerufen und damit der Anwendung beigefügt.

```
function wheelZoom(){
    var navControl = new OpenLayers.Control.Navigation({ // navigation control
     type: OpenLayers.Control.TYPE_TOGGLE,
         zoomWheelEnabled: true});
    map.addControl(navControl); navControl.activate();}
```

Beispiel 23 Implementierung des Mausrades als Zoomwerkezeug

13.3.2 Layertree

Der Layertree ist eine Baumansicht aller anzeigbaren Layer. Diese Form der Darstellung hat sich in verschiedenen GIS-Softwarelösungen durchgesetzt und dient auch dem RFH zum Verständnis der Layerordnung. Der Layertree wird durch eine eigene Funktion in die *index.html* integriert, welche sich selbst in der JavaScriptdatei *layertree.js* befindet.

Innerhalb der Layertreefunktion findet ein einfaches Muster Anwendung. Es wird ein Array[60] erzeugt, indem die Ordner und Layer definiert werden. Ordner können nach Inhalt sortierte Layer beinhalten. Diese Variante der Layerkategorisierung verleiht dem Layertree eine feste Struktur und ermöglicht so das schnelle Finden von Layern. Sowohl für die Ordner als auch die Layer wird über das Attribut *text* der in der Übersicht erscheinende Name definiert. Bei der Festlegung der Ordner wird, zusätzlich zum Namen, mit dem Parameter *expanded* voreingestellt ob dessen Layer direkt auszuwählen sindt (*true*) ist oder nicht (*false*). Zusätzlich wird einem Ordner über das Attribut *children* ein Array beigefügt indem Layer und

[60] ist ein Datenfeld, dient der Speicherung von Daten

neue Ordner definiert sein können. Während der Parameter *expanded* nur für die Ordner zuständig ist, ist der Parameter *checked* sowohl für Ordner als auch Layer zuständig. Dieser gibt an ob ein Ordner bzw. Layer bereits beim Programmstart angezeigt (*true*) wird oder nicht (*false*). Dem Layerobjekt wird über den Parameter *layer* der anzuzeigende Layer zugewiesen. Dabei ist der Layername anzugeben, der bei der Layerdefinition verwendet worden ist. Soll von einer bestimmten Menge an Layern immer nur ein Layer aktiviert sein, so müssen diese über den Parameter *toggleGroup* einer einheitlichen Gruppierung zu geordnet werden. In der Anwendung erscheinen, dann statt Checkboxen, Radiobuttons zur Layeraktivierung. Bei der Verwendung von Radiobuttons ist darauf zu achten, dass ein leerer Layer integriert wird, da eine Deaktivierung eines *toggleGroup* Layers nicht ohne weiteres möglich ist.

Benutzt wird der Layertree ähnlich wie der Microsoft Explorer. Mit einem Klick auf das kleine Plus vor einem Ordnersymbol wird ein Ordner geöffnet oder geschlossen. Layer werden sichtbar, indem den Namen voranstehende, Checkboxen oder Radiobuttons aktiviert werden. Die Reihenfolge und Ordnung der Layer kann durch ein langes Anwählen des Layer- oder Ordnernamens mit der linken Maustaste und anschließenden Ziehens verändert werden. An der neuen Position wird die linke Maustaste einfach losgelassen. Mit dieser Methode können Layer über andere Layer gelegt werden oder umgekehrt.

Der Layertree des RFHs beinhaltet 14 Layer. Die drei Hauptlayer für die Gebäude, Räume und Flurstücke sind direkt über die Hauptansicht verfügbar. Dies ist bewusst so gemacht worden, da die Anwendung für diese drei Layer geschrieben worden ist und so der Zugang zu diesen Dateien direkt und schnell erfolgen kann. Die Hauptlayer können über Checkboxen aktiviert werden, weil nicht immer nur eine Elementkategorie auszugeben ist. Die weiteren Layer unterteilen sich in die Kategorien Hintergrund und Infrastruktur. In dem Ordner Hintergrund ist immer nur ein Layer über Radiobuttons wählbar, weil auch immer nur ein Hintergrund zu sehen sein soll. Aus dem Infrastrukturordner sind die Layer über Checkboxen anwählbar, wodurch alle Layer, beziehungsweise nur ausgewählte Layer oder kein Layer angezeigt werden kann.

```
text: "Infrastruktur",
        expanded: false,
        checked: false, children:[{     text: "Trinkwasser",
                        checked: false,
                        layerName: "trinkwasser"}
```

Beispiel 24 Ausschnitt des Layertrees

13.3.3 Objektliste

Die Objektliste wird im unteren Bildschirm drittel angezeigt und dient der tabellarischen Ausgabe aller verfügbaren Räume, Gebäude und Flurstücke. Ausgegeben werden die einzelnen Objekte mit ausgewählten, dazugehörigen Attributwerten. Die Auflistung ist für alle drei Objektarten verschieden, so dass jede Objektart eine eigene Tabelle hat und zwischen denen hin und her geschaltet werden kann.

Die Objektliste bringt den Vorteil mit das beim Auswählen eines Objektes, sowohl auf der Karte, als auch in der Tabelle, das Pendant farblich hervorgehoben wird.

Die Objektliste wird in der Funktion generiert, in der der dazugehörige Layer erzeugt wird. Demzufolge befinden sich die notwendigen Definitionen in der *wfs_layer.js* Datei. Um einem Layer und einer Tabelle Objekte hinzuzufügen, muss zu erst ein sogenannter Featurestore erzeugt werden. Ein Featurestore ist dabei eine Ansammlung von Daten. Eine solche Datensammlung wird durch die GeoExt Funktion *GeoExt.data.FeatureStore()* an eine Variable gebunden. Innerhalb dieser Funktion wird der datenanzeigende Layer durch den Parameter Layer definiert. Weil die Daten von einem anderen Server, in diesem Falle dem Geoserver kommen, muss ein Proxy mit einem passenden Abfrageprotokoll festgelegt werden. In einem solchen Protokoll wird der Verbindungslink, das Koordinatensystem, der Featuretype, der Feature Namespace und ein Filter angegeben. Der Filter ist anfänglich leer und wird erst durch die Suchfunktion ausgefüllt.

Durch diese Definition werden die Objekte bereits durch den Layer ausgegeben, aber erscheinen noch nicht in Tabellenform. Zur Umsetzung muss einer weiteren Variable eine Funktion angehängt werden, in diesem Fall die *Ext.grid.GridPanel()*-Funktion. In dieser Funktion wird definiert aus welchem Featurestore die anzuzeigenden Daten geladen und welche Daten aus dem Featurestore übernommen werden sollen.

Damit die Tabellen in der Anwendung angezeigt werden, müssen die Variablen als Items einem Viewportobjekt zugeordnet werden.

13.3.4 Suche nach Objekten mit bestimmten Eigenschaften

Die Suchfunktion dient der Ausgabe von Räumen, Gebäuden oder Flurstücken, welche bestimmte Anforderungen erfüllen.

Im Bereich Werkzeuge kann die Suche durch einen einfachen Mausklick auf das sich neben dem Wort *Suche* befindliche Plus geöffnet werden. Es erscheint eine Auswahlliste in der zwischen Gebäude, Räume und Flurstücke gewählt werden kann. Mit der Auswahl erscheint eine Eingabemaske mit den Objektspezifischen Suchkriterien und die entsprechende

Objektliste wird im unteren Bildschirmbereich aktiviert. Für die Kriterien, die die Suche einschränken sollen, werden die Wunschwerte eingegeben. Bei den nicht zuberücksichtigen Kriterien bleibt das Wort Eingabe in den Eingabefeldern stehen. Die mit *suchen* beschriftete Schaltfläche startet die Filterung nach den gesuchten Eingabewerten. Nach wenigen Sekunden haben sich die Objektliste und der entsprechende Layer an die neuen Gegebenheiten angepasst. Weiterhin wird der neu entstanden Kartenausschnitt heran gezoomt, so dass alle Objekte im Kartenfenster zu sehen sind. Mit der Schaltfläche *alle Räume* (bzw. alle Gebäude oder alle Flurstücke) werden die entsprechenden Layer auf den Ausgangszustand zurück gesetzt und alle zugehörigen Objekte werden wieder angezeigt.

Eine solche Suchfilterung wird durch sieben Funktionen ermöglicht. Diese befinden sich in der JavaScriptdatei *suche.js*. Mit der Funktion *suchwechsel()* beginnt die Filterung der Objekte.Jene Funktion sorgt jedoch erstmals nur, für die an die Objekte angepasste Ausgabe einer Eingabemaske. Das wird erreicht, indem ein leeres HTML-Div, mit einem entsprechend, dem ausgewählten Objekt angepassten Div ausgetauscht, wird.

Die eigentliche Suchfunktion ist die Funktion *such()*. In erster Instanz liest diese Funktion, durch die Nutzung der Funktionen *einlesenproperty()* und *einlesenvalue()*, die Suchkriterien und die eingegeben Werte ein. Es werden hierbei nur die Paar beachtet, bei denen das Eingabefeld nicht mehr mit dem Wort Eingabe gefüllt ist. Im nun folgenden Schritt wird die Objektliste in Abhängigkeit des gewählten Objektes (Räume, Gebäude oder eben Flurstücke) gemäß der Eingabewerten gefiltert und neu ausgegeben. Dazu sind die Funktionen *auswertung()* und *filterid()* notwendig. Die Funktion *auswertung()* setzt und erweitert für jedes Property-Value-Paar den Filter durch die Funktion *filterid()*. Nachdem für alle Paare der Filter gesetzt wurde, wird die Objektliste neugeladen und mit ihr der Layer den neuen Parametern angepasst.

Mit *objektlayername.getDataExtent()* wird der neue maximale Kartenausschnitt des gefilterten Layers ermittelt und mit der Funktion *map.zoomToExtent()* an das Mapobjekt übergeben. Dadurch wird die Anzeige aller neu gefilterten Objekte im Kartenbereich ermöglicht. Die Zoomfunktion wird erst ausgeführt, wenn die neue Objektliste vollständig geladen ist, weil sonst die Funktion *getDataExtent()* einen Nullwert erhält und diese zu einem Programmabsturz führt.

Die Suchfunktion findet über eine entsprechende Layerfilterung statt, weil dadurch die Ergebnisse längerfristig, als es bei der normalen Suche über das Highlighting[61] einzusehen

[61] Objekte ändern ihre Farbe und fallen dadurch im Vergleich zu gleichartigen Objekten auf

sind. Außerdem wird durch diese Art der Suche die Objektliste beschränkt, wodurch die Übersichtlichkeit erhalten bleibt.

13.3.5 Drucken

Die Druckmöglichkeit erlaubt es dem Anwendungsnutzer PDF-Dateien zu erstellen auf denen Kartenausschnitte mit einem Titel und Kommentaren versehen werden können.

Die Druckfunktion ist keine selbst geschriebene Funktion. Diese benötigt aber erhöhten Konfigurationsaufwand, weil die Einrichtung eines Proxyservers notwendig ist. Als Proxy dient der Tomcatserver, weil dieser bereits für den Geoserver verwendet wird und somit bereits installiert ist. Zur Einrichtung wird im ersten Schritt die print-servlet-1.1.war Datei in den webapps-Ordner des Tomcatservers kopiert. Weiterhin wird Tomcat mit dem Webserver verbunden. Dazu wird die *htppd.conf* des Webservers mit dem in Beispiel 26 verwendeten Code ergänzt.

```
ScriptAlias /cgi-bin/ C:/ms4w/apache2/cgi-bin/
   <Directory "C:/ms4w/Apache/cgi-bin">
        AllowOverride None
        Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
        Order allow,deny
        Allow from all
   </Directory>
```

Beispiel 25 Einzubindene Zeilen in die htppd.conf des Webservers

Konfiguriert wird die Druckoption über die *config.yaml* Datei im entsprechenden webapps-Verzeichnis des Tomcatdienstes. Hier werden die Zugangsberechtigungen für die Kartendienste erlaubt, die verwendbaren Maßstäbe und Druckauflösungen festgelegt, sowie weitere Layouteigenschaften definiert. Bei der Wahl der importierten Kartendienste ist darauf zu achten, dass diese Dienste auch das Drucken erlauben, weil sonst das Programm abstürzen kann. Ob RFH-externe Dienste gedruckt werden können, ist in den meisten *GetCapabilities*-Dokumenten nachzulesen. Nach dem Ändern der *config.yaml* Datei muss der Tomcatserverdienst neugestartet werden.

Die Integration der Druckmöglichkeit in den Programmcode ist wesentlich einfacher handzuhaben, als die vorhergehende Konfiguration. Im ersten Schritt wurde in der Hauptdatei der RFH-Anwendung der Proxyhost und der Ort der *info.json* an jeweils eine Variable übergeben und eine weitere Datei mit Layereigenschaften versehen.

```
OpenLayers.ProxyHost = "/cgi-bin/proxy.cgi?url=";
printConfigUrl = "http://localhost:8080/print-servlet-1.1/pdf/info.json";
var layerOverrides={ "OpenLayers WMS": {overview: true},Countries: { format: 'image/svg+xml' }};
```

Beispiel 26 Ergänzungen der index.html für die Druckfunktion

Im letzten Schritt wird die viewport-Variable an geeigneter Stelle um die Parameter *x-type:"print-simple"*, der *configURL* und dem Objekt *overrides* ergänzt.

```
{ xtype: 'print-simple',
  bodyStyle: 'padding: 7px;',
  border: false,
  map: map,
  configUrl: printConfigUrl,
  overrides: layerOverrides  }
```

Beispiel 27 Erweiterung des Viewports durch die in Bsp. 26 definierten Variablen

Um die Druckmöglichkeit im RFH zu nutzen, muss in der Werkzeugleiste der Anwendung das Drucken aktiviert werden. Indem das *Plus* neben dem Wort *Drucken* betätigt wird. Daraufhin erscheint in der Werkzeugleiste eine neue Eingabemaske und in der Karte ein oranges, leicht transparentes Rechteck mit einem ebenfalls orangen, kleinem Kreis auf einer der Rechteckkanten. In der Eingabemaske können nun Maßstab und Druckauflösung eingestellt werden und der PDF Datei ein Titel und Kommentare beigefügt werden. Das Rotieren, des zu erstellenden Dokumentes, kann über die Eingabe eines Rotationswinkels oder über das Ziehen des kleinen Kreises mit der gedrückten, linken Maustaste erfolgen. Das Verschieben des Rechteckes passiert ebenfalls über die gedrückte, linke Maustaste und anschließender Bewegung in die gewünschte Position. Sind alle Einstellungen den Wünschen entsprechen getätigt worden, so wird über die Schaltfläche *print* eine PDF-Datei generiert. Diese Datei kann nun ausgedruckt, gespeichert oder zum Beispiel als Email versendet werden.

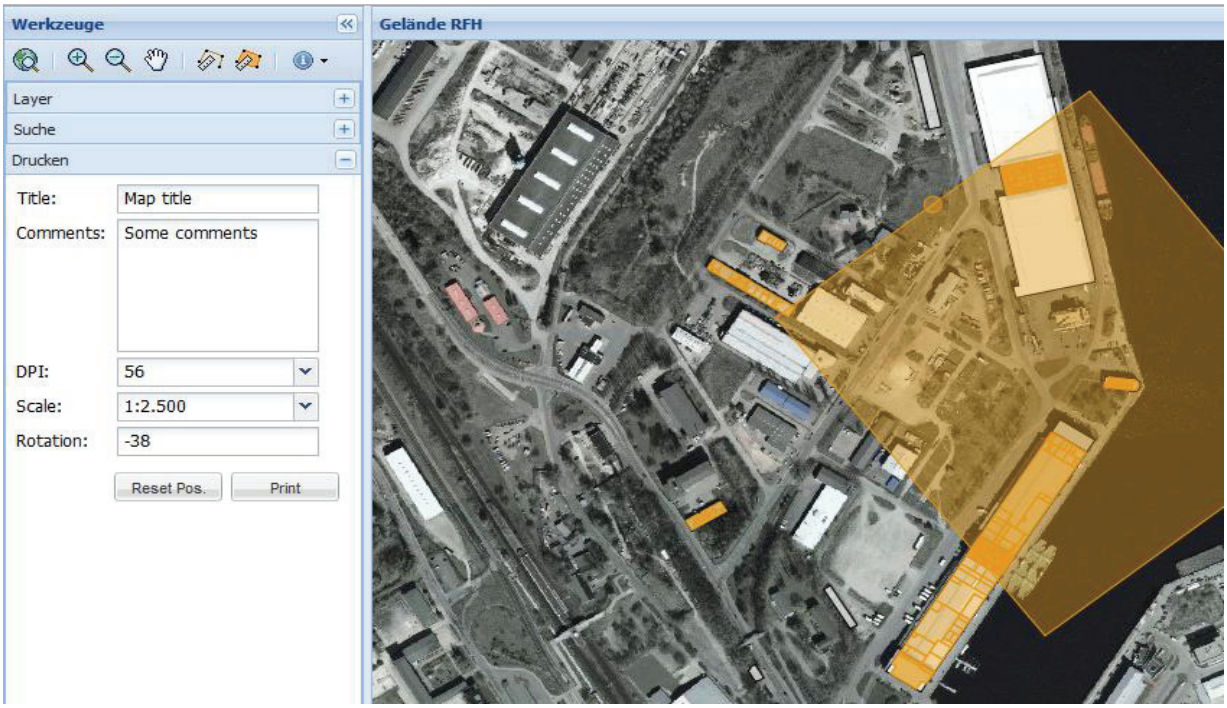


Abbildung 25 Anwendung im Druckmodus

13.3.6 Toolbar

Mit Toolbar wird eine Werkzeugleiste bezeichnet, mit der über Schaltflächen, Funktionalitäten bereitgestellt werden. Eine solche Art der Funktionsbereitstellung verhindert unnötiges, langes Suchen von Funktionen und ist auch in anderen Programmen, außerhalb der GIS-Welt, wie z.B. in Microsoft Word etabliert. Die Werkzeugliste des RFHs beherbergt die in Tabelle 8 aufgeführten Funktionen.

Funktion	Beschreibung	OpenLayers.Control.
Gesamtausschnitt	Kartenausschnitt wird auf das Maximum gezoomt	ZoomToMaxExtent()
Vergrößern	Kartenausschnitt wird vergrößert	ZoomBox()
Verkleinern	Kartenausschnitt wird verkleinert	ZoomBox({out:true})
Distanzmessung	Ausgabe einer Streckenlänge in m	Eigene Funktion
Flächenmessung	Ausgabe einer Flächengröße in qm	Eigene Funktion
Informationen	Ausgabe von Informationen zu einem angewählten Objekt	SelectFeature()+eigene Funktion

Tabelle 8 Funktionen der Toolbar

Die Aktivierung, einer der der acht Schaltflächen, erfolgt durch einen einfachen Klick der linken Maustaste auf die Schaltfläche selbst. Auf demselben Weg lassen sich die Funktionen auch deaktivieren. Eine Deaktivierung der vorher genutzten Funktion ist unnötig, weil dies durch die Nutzung einer anderen Funktion automatisch geschieht. Ist eine Funktion wirklich

aktiv, ist die dazugehörige Schaltfläche in der Werkzeugliste reliefartig, tiefer gesetzt. Nach der erfolgreichen Aktivierung eines Werkzeuges, kann dieses im Kartenfenster benutzt werden. Eine Ausnahme bildet die Benutzung des Weltwerkzeuges (MaxExtent-Funktion), denn diese Funktion wird direkt mit der Schaltflächenaktivierung ausgeführt. Bei den beiden Funktionen Vergrößern und Verkleinern kann mit der linken Maustaste ein rotes Rechteck über den gewünschten Ausschnitt aufgezogen werden oder ebenfalls mit der linken Maustaste einfach in den Kartenbereich geklickt werden, um den gewünschten Effekt zu erzielen. Auf die Nutzung und Implementierung der Distanz- und Flächenmessung, sowie der Ausgabe von Objektinformationen wird explizit in den beiden folgenden Abschnitten eingegangen.

Implementiert wird die Toolbar über eine eigene Funktion, die in der *index.html* aufgerufen wird und in der Datei *toolbar.js* hinterlegt ist. In der Funktion *creatItems()* wird ein Array mit den Werkzeugobjekten angelegt um die Toolbar zu erzeugen.

Eine Funktion erhält ein Werkzeugobjekt über den Parameter control, das heißt es wird angegeben, welche Funktion bei aktiver Schaltfläche ausgeführt werden soll. Der zweite zu definierende Parameter gibt an, auf welches Objekt sich die Funktion bezieht (hier ist es das Mapobjekt). Weitere Eigenschaften eines Werkzeugobjektes sind die Iconclass (=iconCls), die dem Objekt ein Schaltflächenbild zuweist, und der Tooltip, der durch Mauszeigerberührung mit der Schaltfläche einen Nutzungshinweis ausgeben kann. Damit auch wirklich nur eine Funktion aufrufbar ist und somit keine Funktionsüberschneidung zu Programmkomplikationen führen kann wird den Werkzeugobjekten eine einheitliche *toggleGroup* zugewiesen. Sind alle Eigenschaften festgelegt worden wird das Werkzeug mit der JavaScriptfunktion *push()* in das Array eingefügt.

Die Ausgabe der Schaltflächen erfolgt exakt in der Reihenfolge, wie die Funktion in dem Array hinterlegt worden sind. Das heißt, dass das zu erst gespeicherte Objekt in der Toolbar ganz links ausgegeben wird und dass das zu letzt abgespeicherte Objekt demzufolge ganz rechts.

```
action = new GeoExt.Action({
        control: new OpenLayers.Control.ZoomToMaxExtent(),
        map: map,
        iconCls: 'zfull',
        toggleGroup: 'map',
        tooltip: 'Gesamtkartenausschnitt'});toolbarItems.push(action);
```

Beispiel 28 Werkzeugobjekt für die Toolbar

13.3.6.1 Distanz- und Flächenmessung

Die Werkzeuge Distanz- und Flächenmessung dienen der Bestimmung von Strecken- und Teilstreckenlängen und der Ausgabe von Flächengrößen.

Die Integration dieser beiden Werkzeuge, in die Anwendung, erfolgt über die Toolbar, welche auch das Starten der Funktionen ermöglicht. Ist eines der beiden Werkzeuge aktiviert kann die Messung ausschließlich im Kartenbereich stattfinden.

Bei der Linienmessung wird beim ersten Klick mit der linken Maustaste der Startpunkt der Messung festgelegt und mit jedem weiteren Klick wird eine weite Teilstrecke der Messung hinzugefügt. Abgeschlossen wird die Distanzmessung durch einen Doppelklick auf die gewünschte letzte Position.

Die Messung einer Fläche basiert auf demselben Prinzip. Beim ersten Klick wird, ebenfalls mit der linken Maustaste, der erste Flächeneckpunkt gesetzt. Mit jedem weiteren Klick wird das Polygon um einen Eckpunkt erweitert. Auch hier wird mit einem Doppelklick das Ende der Messung bewirkt. Bei beiden Messungsfunktionen öffnet sich beim ersten Mausereignis, im rechten Bildschirmbereich, das Informationsfenster, in dem die Streckenlängen in Meter und die Flächengröße in Quadratmetern ausgegeben wird.

Die Messungsfunktionen werden in der eigens angelegten Datei *messen.js* definiert und als *control* an den zuständigen Werkzeugobjekten der Toolbar verwendet. Doch bevor diese Funktionen richtig arbeiten können, muss die Präsentationsanwendung noch erweitert werden. Die erste Erweiterung findet im Viewport der *index.html* Datei statt. Im east-Panel wird dafür, über den Parameter *html*, ein Platzhalter mit dem Namen *information* definiert. Dieser Platzhalter ist für die Ausgabe der Messungsergebnisse zuständig. Eine weitere Erweiterung ist die Integration und Vervollständigung der proj4js-Bibliothek in die Anwendung. Diese JavaScript-Bibliothek sorgt für die Umwandlung, der durch die Verwendung des EPSG-Codes 2398 bedingten geodätischen Einheiten in ein metrisches System. Die Bibliothek wurde einfach in den Clientordner der Anwendung kopiert und in die *index.html* eingebunden. Weil die proj4js-Bibliothek nicht von vornherein den EPSG-Code 2398 unterstützt, muss für diesen Code eine separate JavaScriptdatei mit den Eigenschaften angelegt werden. Diese Eigenschaften sind in Beispiel 29 zu sehen.

```
Proj4js.defs["EPSG:2398"]="+title=Pol, LCC +proj=tmerc +lat_0=0 +lon_0=12 +k=1.000000 +x_0=4500000 +y_0=0 +ellps=krass +towgs84=24,-123,-94,0.02,-0.25,-0.13,1.1 +units=m +no_defs";
```

Beispiel 29 Eigenschaften des Pulkovkoordinatensystems

Nachdem diese Vorbereitungen abgeschlossen sind, steht der Umsetzung der in *messen.js* definierten Funktionen, nichts mehr im Wege. Im ersten Schritt werden über eine Variable die Messungseigenschaften definiert. Das bedeutet hier wird unter anderem festgelegt, dass das Ausgangskoordinatensystem geodätisch ist. Daraufhin wird für jede Messungsart eine Controllervariable definiert, die bei der Erzeugung der Werkzeugleiste dem jeweils zuständigen Werkzeugobjekt als *control*-Parameter mitgegeben wird. Dieser Variable werden, über die Openlayers Measurefunktion, die Messungseigenschaften und eine Zeichenfunktion für die Linien bzw. die Polygone übergeben. Weiterhin wird festgelegt welche Funktion, während des Zeichnens der Linie bzw. des Polygons ausgeführt werden soll.

```
var measureOptions = {
persist:true,
geodesic:true,
handlerOptions : {
style:"default"}          }
var lineMeasure = new OpenLayers.Control.Measure(OpenLayers.Handler.Path,
measureOptions);
var polygonMeasure = new OpenLayers.Control.Measure( OpenLayers.Handler.Polygon,
measureOptions);
lineMeasure.events.on( {
"measure" :handleDistanceMeasurements,
"measurepartial" :handleDistanceMeasurements      });
polygonMeasure.events.on( {
"measure" :handleAreaMeasurements,
"measurepartial" :handleAreaMeasurements});
```

Beispiel 30 Ausschnitt aus den Messungsfunktionen

Während des Zeichnens eines Polygons wird die Funktion *handleAreaMeasurements()* ausgeführt. Diese reagiert bei jedem Mausereignis und berechnet dadurch die Fläche aus den definierten Eckpunkten. Das Ergebnis wird im Informationsfenster ausgegeben.

Für die Berechnung der Strecke- und Teilstrecken ist die Funktion *handleDistanceMeasurements()* verantwortlich. Prinzipiell arbeitet diese Funktion genauso wie die der Fläche, jedoch mit dem Unterschied dass auch Teilstrecken ausgegeben werden sollen. Daher wird im ersten Schritt der Informationsplatzhalter mit den drei Platzhaltern *messung, teilmessung* und *status2* gefüllt. Das Div *messung* gibt die gesamte Streckendistanz

aus, im Platzhalter *teilmessung* werden die Teilstrecken angezeigt und über *status2* wird gespeichert, ob das zuletzt ausgegebene Teilstück ein Endstück war oder nicht.
Die Berechnung der Teilstrecken wird erzielt, indem alle Teilstrecken von der aktuellen gesamt Distanz subtrahiert werden. Die Differenz wird im Platzhalter *teilmessung* angehängt und mit den restlichen Teilstrecken ausgegeben. Hat die Teilstrecke den Wert null, so gilt die Messung als abgeschlossen. Weil nur bei einem Doppelklick wird ein und dieselbe Koordinate zweimal getroffen. Durch dieses Ereignis wird *status2* auf „2“ gesetzt, was zur Folge hat, dass bei einer erneuten Streckenmessung alle Teilmessungen gelöscht werden und diese bei null beginnen.

13.3.6.2 Information über einzelne Objekte

Die Funktion zur Ausgabe von Objektinformationen, über den Informationsbereich dient der Anzeige von Informationen die eventuell nicht in den Objektlisten vorkommen. Als wesentliche Informationsergänzung werden Bilder der selektierten Objekte mit ausgegeben. Dadurch kann ein persönlicher Abgleich stattfinden, wie zum Beispiel die Kontrolle, ob das gewählte Objekt auch das gewünschte Objekt ist.
Um Informationen zu einem Objekt im Informationsbereich auszugeben, wird in der Werkzeugleiste eine von drei mit *i* gekennzeichneten Schaltflächen betätigt. Die Bebilderung der Schaltflächen gibt dabei an zu welcher Objektart die Informationen ausgegeben werden sollen. Das *i* mit einem *R* im Hintergrund aktiviert die Ausgabe der Rauminformationen, das *i* mit einem *G* im Hintergrund aktiviert die Ausgabe der Gebäudeinformationen und das *i* mit einem *F* im Hintergrund aktiviert die Ausgabe der Flurstückinformationen. Mit der Aktivierung, einer dieser Schaltflächen, wird der abzufragen Layer in der Kartenhierarchie in den Vordergrund gestellt. Wodurch das Abfragen bestimmter Objekte erst ermöglicht wird. Durch einen einfachen Mausklick auf das gewünschte Objekt wird dieses selektiert und die dazugehörigen Attribute werden in dem aufgegangenen Informationsbereiche ausgegeben.
In die Präsentationsanwendung werden die Informationsschaltflächen über die Toolbar integriert. Dazu wird den zuständigen Werkzeugobjekten die Funktion *OpenLayers.Control.SelectFeature(),* inklusive dem abzufragenden Layer und einer Funktion, die besagt was bei der Objektselektion geschehen soll, übergeben.

```
action = new GeoExt.Action({
        iconCls: 'info',
        tooltip: 'map',
        control: new OpenLayers.Control.SelectFeature(gebaeude, {
        type: OpenLayers.Control.TYPE_TOGGLE,
        onSelect: onFeatureSelectGebaeude}),
```

Beispiel 31 Informationswerkezug für die Toolbar

In der Datei *info.js* sind die zusätzlichen Funktionen definiert. Über die Controlfunktion des Werkzeugobjektes gelangt die Anwendung an das selektierte Objekt. Dieses wird im Programmcode als Feature bezeichnete und an die selbstgeschriebenen Funktionen übergeben. Innerhalb der eigenen Funktionen ist festgelegt, dass der Informationsbereich sich zu öffnen hat, sofern er dieses noch nicht gemacht hat.

```
Ext.getCmp('east').expand();
```

Beispiel 32 Funktion zum Ausklappen des Informationsbereichs

Gleichzeitig wird der im Informationsbereich vorhandene Platzhalter, um weitere Div-Elemente aufgefüllt. Jedes neu hinzugefügte Element dient der Ausgabe eines Objektattributes. Dieser Umstand wird erreicht, indem über *document.getElementById('Divname').innerHTML='Wert'* jedem Platzhalter ein Wert zugewiesen wird. Damit die Werte auch den Attributen des selektierten Wertes entsprechen, wird über *feature.data.+Attributspaltenname* das entsprechende Objektattribut ausgegeben. Für die auszugebenden Bilder wird der HTML-Code bereits vorgefertigt, damit an die Bildquelle nur noch der Name gehängt werden muss.

```
Einem Divelement wird ein Featueattribut übergeben:
document.getElementById('gebaeudebezeichnung').innerHTML=feature.data.bezeichnung;
Bildquelle:
<img src="images/'+feature.data.bild+'" width="100" height="100" border="0" alt="Gebaeude">
```

Beispiel 33 Zusammenspiel von JavaScript und HTML zur Ausgabe von Objektattributen

13.4 optische Struktur

Die Anordnung der einzelnen Präsentationselemente ähnelt bewusst derer bekannter GIS-Lösungen, wie ArcView oder Google Earth, um anfängliche Orientierungsschwierigkeiten zu vermeiden. Damit die Präsentationsplattform nicht zu verspielt oder altmodisch wirkt, wurde

die voreingestellte, schlichte und elegante Farbwahl der Ext-Entwickler übernommen. Auch hier erfährt der Nutzer einen Wiedererkennungswert, denn die farbliche Gestaltung beruht sehr auf der aktuellen Microsoft Office Version.

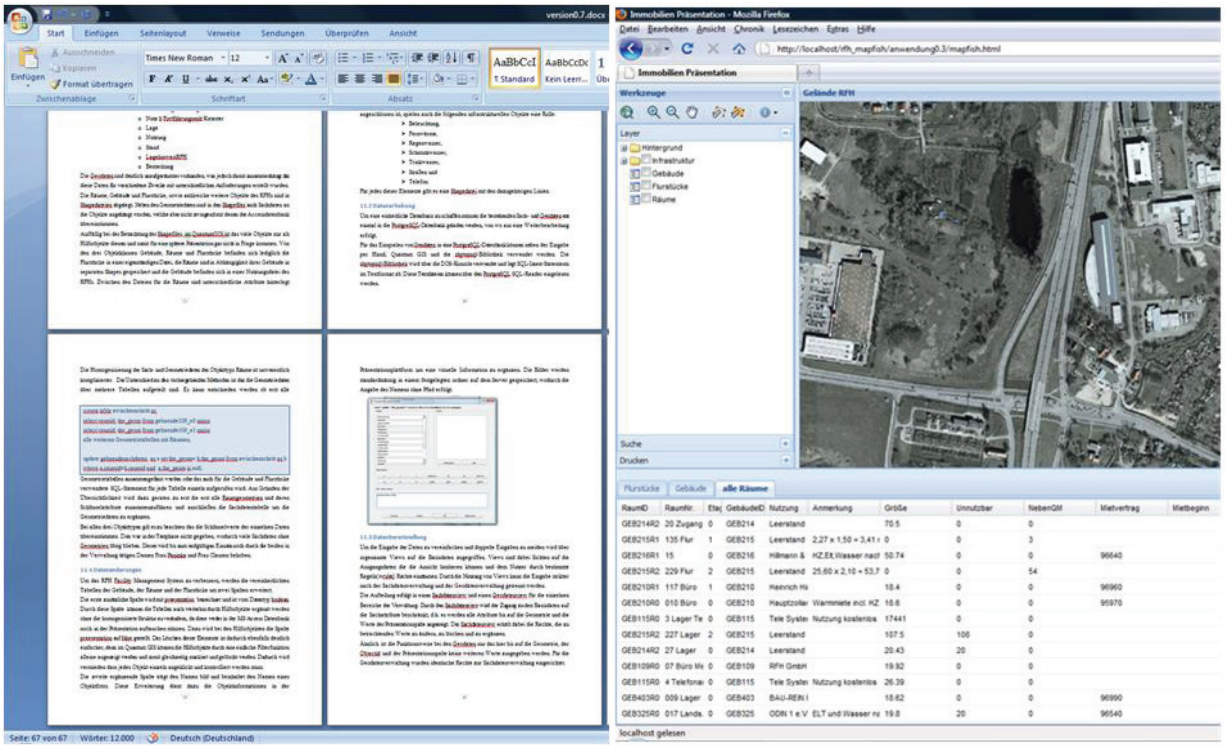


Abbildung 26 Vergleich Word und Mapfish

Gegliedert ist die Webanwendung in vier Bereiche. Diese sind das kartendarstellenden Fenster, das Funktionsfenster, die Objektliste und das optionalen Informationsfenster. Beim Starten der Präsentationsplattform sind vorerst bis auf das Informationsfenster alle weiteren Elemente zu sehen. Das Kartenfenster ist fest verankert und nimmt die meiste Bildschirmfläche in Anspruch. Links neben der Karte befindet sich, deutlich schmaler, die Funktionsleiste. Diese beherbergt eine immer aktive Toolbar, sowie drei aktivierbare Steuerungselemente. Das ist zum einem die Layerübersicht, die Druckfunktion, sowie die Möglichkeit die angezeigten Elemente einzuschränken. Im gesamten unteren Drittel befinden sich drei Tabellen, die die geladenen Räume, Gebäude und Flurstücke samt ihrer Attribute anzeigen. Zwischen den Tabellen kann über Reiter gewechselt werden. Das vierte Fenster erscheint nur bei aktiver Nutzung der speziellen Funktionen aus der Werkzeugleiste. In diesem Fenster werden zusätzliche visuelle Informationen zu den Objekten oder Strecken- und Flächenmessungen ausgegeben.

Außer dem Kartenelement können alle Fenster zur Seite oder nach unten geklappt werden, um mit der Karte den nahezu ganzen Bildschirm zu füllen.

Zur farblichen Gestaltung der einzelnen Layer des RFHs können noch keine Angaben gemacht werden, weil diese erst bei Inbetriebnahme von Seiten des RFHs festgelegt werden.

Die gerade beschriebene Anwendungsoptik wird in der Viewvariable in der *index.html* festgelegt. Ihr wird die aus der Ext-Bibliothek stammente Funktion *Ext.Viewport* übergeben. Die Funktion beinhaltet ein Array, welches alle auszugebenden Elemente fasst. Über Parametern können die Elemente Eigenschaften annehmen und vorkonfiguriert werden.

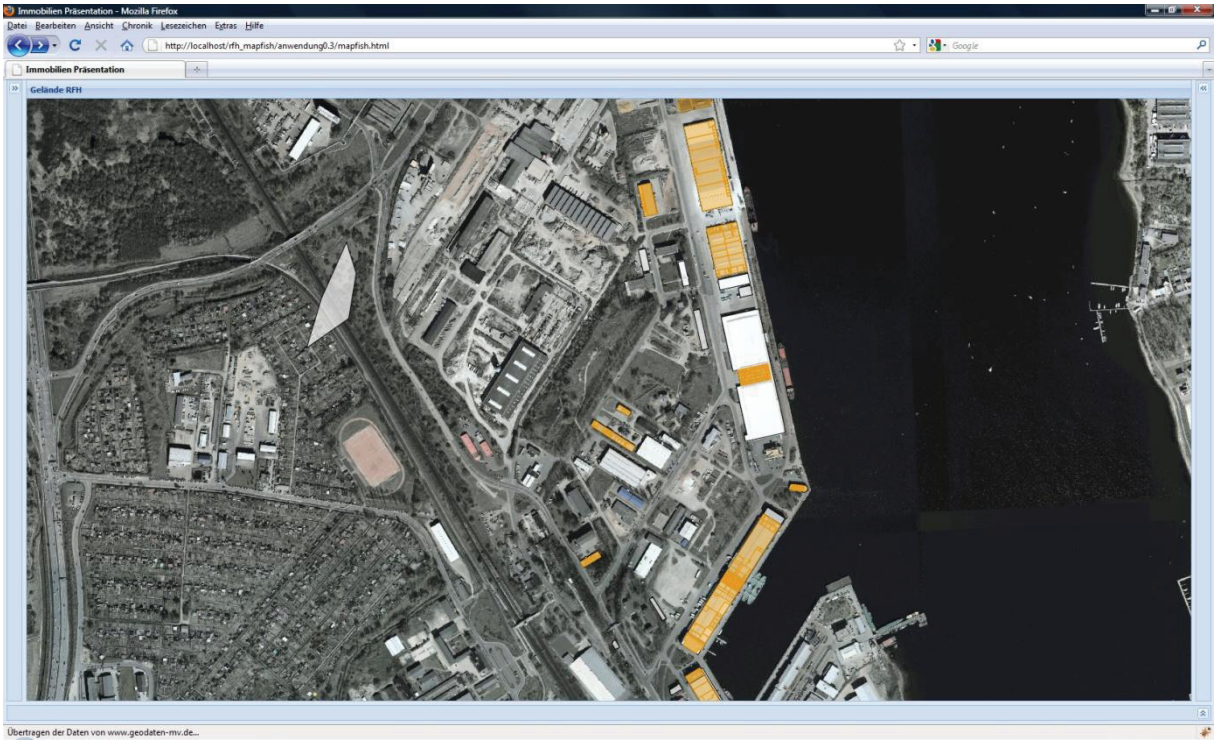


Abbildung 27 Ansicht mit eingeklappten Leisten

Fazit

14 Zusammenfassung

Die neugeschaffenen Strukturen des Facility Management Systems im Rostocker Fracht- und Fischereihafen erschaffen, neben der Datenpräsentationsplattform auch effektivere und freundlichere Arbeitsverhältnisse. Die Änderungen sind dabei hauptsächlich im Hintergrund vorgenommen worden, wodurch nur geringe Umstellungen der Arbeitsweisen erfolgen müssen.

Das Fundament der strukturellen Veränderungen legt die Postgresql-Datenbank zusammen mit dem Geodaten unterstützenden Aufsatz PostGIS. Diese Datenbank sorgt für die Zusammenführung der Sachdaten mit den dazugehörigen, geometrischen Attributen. Durch das Anlegen von Sichten (in der Postgresqlsprache als Views bezeichnet), mit bestimmten Regeln, können diese Daten nur von befugten Mitarbeitern ergänzt und geändert werden. Mit der einheitlichen Datenbasis sind die zu verwaltenden Daten, sowohl im neu eingefügten Quantum GIS und in der Microsoft Access Datenbank, immer auf dem aktuellen Stand. Dieser Fakt verhindert die Mehrfachspeicherung der Objekte, weil für ein Objekt nicht ein Sachdatensatz und ein Geodatensatz angelegt werden muss. Bei dieser Art der Datenverwaltung ist lediglich darauf zu achten, dass die Objekte eindeutige Identifikationsschlüssel zugewiesen bekommen. Hierbei sind eventuell Absprachen zwischen den Verwaltungsmitarbeiterinnen zu tätigen.

Damit die Mitarbeiter der RFH GmbH nicht von der Verwaltungssoftware und den Fachkenntnissen, der in der Verwaltung tätigen Kollegen abhängig sind, wurde ein internetbasiertes GIS-System entwickelt. Dieses wurde als datenpräsentierende Ausgabeplattform im RFH integriert. Dieses GIS-Tool läuft über den hauseigenen Server und kann, über jeden an das Intranet angeschlossenen Arbeitsplatz, mit einem installierten Firefox oder Chromebrowser gestartet werden. Eine extra Installation auf den Arbeitsplatzcomputern ist nicht notwendig. Die Plattform bietet alle vom RFH gewünschen Funktionen in einem optisch ansprechendem Gewandt. Neben den Standardfunktionen, wie der Navigation, dem Vergrößern und Verkleinern von Kartenausschnitten enthält die auf Mapfish basierende Softwarelösung Funktionen, die die Messung von Distanzen und Flächen erlauben. Die Ausgabe von gezielten Objektinformationen inklusive von Objektbildern ist wunschgemäß implementiert worden. Jeder Nutzer hat zudem die Möglichkeit die Ausgabe der Räume, Gebäude und Flurstücke über einen Suchfilter einzuschränken und so an seine Bedürfnisse anzupassen. Weiterhin obliegt es jedem Mitarbeiter seine aktuellen Kartenansichten mit verschiedenen Optionen über das Generieren von Pdf-Dokumenten zu speichern oder auszudrucken.

Die Erweiterung der Intranetanwendung kann durch einen Mitarbeiter mit Programmierkenntnissen in JavaScript aktualisiert und weiterentwickelt werden. Die aktuelle Version bietet durch die Kommentierung der selbstgeschriebenen Programmcodes eine grundlegende Unterstützung dafür.

Neben den bis dato genannten positiven Aspekten besitzt die Präsentationsanwendung auch zwei bekannte Nachteile. Der am ehesten korrigierbare Fehler ist die nicht gegebene Lauffähigkeit der GIS-Anwendung im Internet Explorer. Diese lässt sich durch die bereits erwähnte Installation des Firefox-Browsers beziehungsweise des Chrome-Browsers oder der Anpassung des JavaScriptcodes beheben. Viel gravierender ist die Tatsache, dass auf Grund der erhöhten Anzahl an Räumen und JavaScriptfunktionen der Programmstart eine längere Ladezeit benötigt. Wird hier geduldig auf das Verschwinden der „Bitte Warten“-Maske gewartet, sollte das Programm durchaus stabil lauffähig sein.

Ein Vorteil, welches alle neu eingeführten Produkte des Strukturwandels im RFH betrifft, ist der Fakt das diese Produkte Open Source-Lösungen mit einer großen Community sind. Durch diesen Umstand ist der kostenfreie Einsatz im RFH gewährleistet, ohne dass Sanktionen, auf Grund fehlender Lizenzen zu befürchten sind. Ein weiterer Faktor für die Einführung dieser Softwarelösungen ist die bereits genannte große Community der einzelnen Produkte, diese bietet kostenlose Hilfe über Foren an. Diese Art der Hilfe ist eventuell nicht ganz so praktisch und kann auch unbefriedigend im Vergleich zur Hilfe einer Hotline sein, hat dafür den Vorteil dass nicht minutenlang am Telefon die Arbeitszeit mit Nichtstun verbracht werden muss. Ein weiteres Zeichen für die Langlebigkeit der verwendete Produkte ist die bereits lange Verfügbarkeit auf dem Softwaremarkt und die Ankündigung neuer Versionen und Updates auf den einzelnen Internetseiten.

15 Ausblick

15.1 Anwendung im RFH

Der Einsatz der Mapfish-Bibliotheken öffnet dem RFH weitere Türen.

Ausgelegt ist die aktuelle Installation als reine Datenpräsentation. Das Zusammenspiel der Verschiedenen Bibliotheken und die grundlegende Implementierung der Funktionen ermöglichen, neben der Datenbetrachtung, auch die Datenbearbeitung und die Datenerweiterung. Microsofts Datenbank Access könnte als Eingabesoftware komplett abgelöst werden. Denn die Sachdaten können, durch einfache Veränderungen des Programmcodes, in den Objektlisten geändert und gespeichert werden. Die Ext JS–Bibliothek bietet sogar zusätzliche, vorgefertigte Formulare für die Einspeicherung und Änderung von Sachdaten an. Dieser Schritt würde ermöglichen, dass die Eingabe von Sachdaten komplett unabhängig vom Arbeitsplatz und unabhängig von Access über das Intranet erfolgen könnte.

Ein weiterer Schritt innerhalb des RFHs ist die Realisierung der Geodatenbearbeitung über die Mapfishanwendung im Browser. Die bereits enthaltenen WFS-Schnittstellen sind durch die aktuelle Implementierung bereits für einen Einsatz von WFS-T vorkonfiguriert. Das heißt über bestimmte Funktionserweiterungen können, zwischen der Anwendung und der Datenbank über den Geoserver Transaktionen, wie der Änderungen, dem Löschen und der Erweiterung der Geodaten durchgeführt werden. Das Zeichnen und Ändern von Geodaten soll nach Angaben der CamptoCamp-Programmier im Browser neue Dimensionen erreichen. Auf der Mapfishprojekthomepage ist zu lesen, dass viele Konstruktionsfunktionen, wie unter anderem das Snapping[62] durch die die neue OpenLayersversion 2.8 fester Bestandteil von Mapfish werden soll. Diese Erweiterungen würden das Desktop GIS als Zeichenwerkezug vollständig ablösen können.

Durch diese möglichen Erweiterungen, könnte die Anwendung nur noch einer Software in Betracht gezogen werden. Auf Access und Quantum GIS wird dann gänzlich verzichtet. Zusätzlich wäre es eine Hilfe bei der Sachdateneingabe, weil die zu ändernen Sachattribute sich nur auf das markierte Objekt beziehen. Eine Absprache zwischen den Verwaltungsbereichen müsste nicht extra stattfinden. Die Eingabe von Sach- und Geodaten könnte durch die Integration eines Loginprotokolls, welches den Nutzern unterschiedliche Rechte und Funktionen zuteilt, weiterhin durch unterschiedliche Bearbeiter stattfinden.

Ein weiter Schritt der die Attraktivität der Präsentationsplattform ungemein steigern würde, wäre die Implementierung eines Statistikmoduls, mit dem die Daten nicht nur durch einen

[62] einfangen von Punkten oder Linien

Filter gesondert ausgegeben werden können, sondern auch Abfragen nach statistische Kriterien ausführt. So wäre eine Ausgabe der prozentualen Raumauslastung oder eine Gegenüberstellung von sanierten und sanierungsbedürftigen Räumen von Interesse. Solche Ausgaben könnten durch Diagramme veranschaulicht werden.

Die immer häufigere Anwendung von Web 2.0-Anwendungen im Internet (StudiVZ oder YouTube) fordert immer schnellere und vor allem kompatiblere Browser. Die Entwicklung der letzten Jahre zeigt einen enormen Geschwindigkeitsgewinn durch die neuen Browser. In diesem Bereich sehen die Browserenwickler ihre Chance neue Marktanteile zu gewinnen und werden auch weiterhin an der Performance ihrer Programme arbeiten. Wodurch die nächsten Browsergenerationen die langen Ladezeiten von XML-Dokumenten und viel JavaScriptcode vergessen lassen werden.

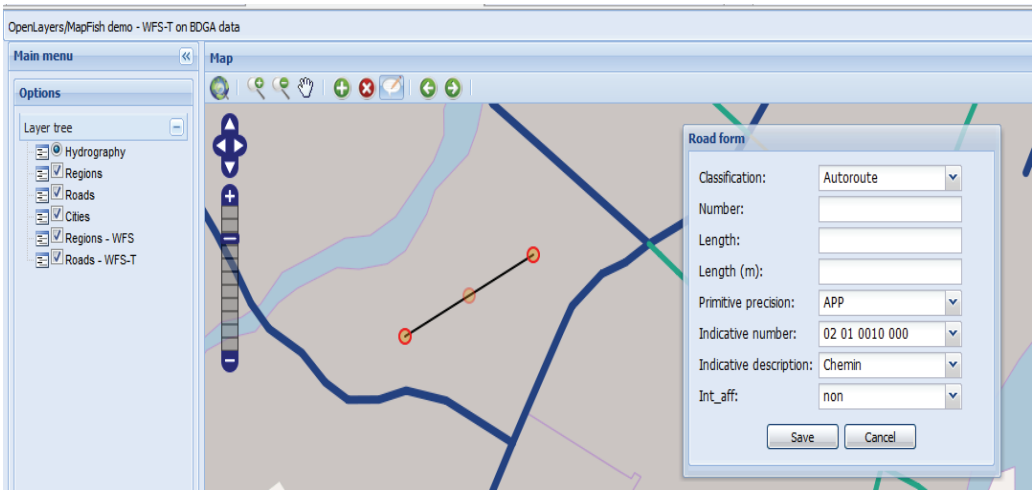


Abbildung 28 Formular zum Ändern von Sachattributen und Änderung der Geometrie eines Objektes

15.2 Anwendungen außerhalb des RFHs

Die Anwendungsstruktur kann durch die zentrale Datenbank auch für andere Unternehmen außer dem RFH mit Geodaten interessant sein. Einige Organisationen, die eigentlich mit Geodaten arbeiten, speichern diese oft nur als einfache Sachdaten, ohne die geometrische Komponente. In vielen Fällen wird der Schritt der Geodateneinführung, durch die Unternehmen, nicht gewagt, weil es an Fachkräften beziehungsweise eigenem geschultem Personal mangelt. Die Trennung der Aufgabengebiete im RFH zeigen, dass die Arbeit der Geodaten- und Sachdatenverwaltung an einem Datensatz erfolgreich sein kann. So können Unternehmen Geodaten verwalten ohne das jeder Mitarbeiter diese Kenntnisse besitzen muss und dennoch das Zusammenspiel mit den Sachdaten beibehalten wird. Viele Immobilienanbieter könnten ihren Angeboten Grundrisse beifügen und den Kunden über das

Internet neben den Sachdaten präsentieren. Bis dato sind die Grundrisse der meisten Immobilien nur als Bild hinterlegt. Eine Hinterlegung der Grundrisse in einer internetbasierten GIS-Anwendung würde zudem weitere Informationen direkt vermitteln können, die eventuell erst bei einer Wohnungsbesichtigung zu Tage kommen würde. Als Beispiel kann die Ausrichtung eines Schlafzimmers angebracht werden. Über den Grundriss mit unterlegter Luftbildaufnahme lässt sich direkt erkennen, ob das Schlafzimmer zur lauten Straße oder ruhigen Innenhof gelegen ist. Zu beachten ist dabei sicherlich, dass viele Immobiliengesellschaften gar nicht über jeden Grundriss verfügen. Eventuell wären diese Unternehmen aber durch eine solche Datenpräsentation gezwungen nach und nach für ihre Objekte Grundrisse aufzunehmen. Diese hätte den Effekt, dass der Mieter auch aktuelle Grundrisse als Information erhält und nicht oftmals sehr alte Zeichnungen.

Eingesetzt werden könnte die Anwendung auch in RFH ähnlichen Verwaltungen, wie einer Einkaufszentrumsverwaltung oder Gewerbeflächenverwaltung, denn diese könnten so ihre Verwaltungsoberfläche mit einer visuellen Komponente füllen und bereichern.

Beim Testen der Mapfishbeispiele konnte das 3D-View Plugin von Google Earth die Aufmerksamkeiten auf sich ziehen. Dieses Plugin ermöglicht die Kombination einer räumlichen 3D-Ansicht und der 2D-Kartendarstellung., über die sich die Kamera des 3D-Fensters steuern lässt. Die genaue Positionierung der Kamera lässt sich durch wenig Variablen exakt vordefinieren und variabel gestallten. Sowohl Touristik- als auch Immobilienunternehmen könnten an solcher Art der Datenpräsentation interessiert sein, denn über wenige Parameter lässt sich der Blick aus einem Hotelzimmer oder einer Wohnung simulieren. Durch das Einbinden von digitalen Stadtmodellen lässt sich nachhaltig Werbung für den unbebauten Meeresblick oder den Blick auf den Park machen.

Insgesamt bietet die Datenstruktur des RFHs mit ihrer zentralen Datenbank und der Mapfishanwendung auch außerhalb des RFHs Perspektiven.

Abbildung 29 3D View mit Mapfish und Google Earth Plugin

Verzeichnise

Literatur- und Quellenverzeichnis

[1] 15221-1, DIN EN. *Facility Management - Teil 1: Begriffe; Deutsche Fassung EN 15221-1:2006.* 2006.

[2] *Aktuelle Browser im Benchmark-Test.* http://www.zdnet.de/bildergalerien_aktuelle_browser_im_benchmark_test_story-39002386-41004997-1.htm#g (Zugriff am 30. 05 2009).

[3] Bartelme, Norbert. *Geoinformatik-Modelle, Strukturen, Funktionen.* Berlin: Springer, 2005.

[4] Burger, Cornelia Boenigk & Ralf. *PostgreSQL, das fortschrittlichste Open Source Datenbanksystem.* 23. 06 2009. http://www.postgresql.de/index.whtml#pg (Zugriff am 23. 06 2009).

[5] CamptoCamp. *Mapfish.* http://trac.mapfish.org/trac/mapfish/wiki (Zugriff am 10. 01 2009).

[6] Cédric. „Alles was Sie mit MapFish machen können.“ http://mapfishblog.blogspot.com/2009/03/alles-was-sie-mit-mapfish-machen-konnen.html (Zugriff am 09. 06 2009).

[7] *Ext JS A foundation you can bild on.* http://www.extjs.com/ (Zugriff am 01. 10 2009).

[8] Fischereihafen, Rostocker Fracht- und. *Rostocker Fracht- und Fischereihafen.* 25. 04 2009. http://www.rfh.de/de (Zugriff am 25. 04 2009).

[9] *Geoserver.* http://geoserver.org/display/GEOSDOC/Operation (Zugriff am 19. 07 2009).

[10] *Google Chrome - Laden Sie einen neuen Browser herunter.* http://www.google.de/chrome/ (Zugriff am 23. 09 2009).

[11] *IE, Windows & Google verlieren Marktanteile.* http://digitalewelt.freenet.de/softwareos/windows/ie-windows--google-verlieren-marktanteile_816228_275572.html (Zugriff am 14. 08 2009).

[12] *Introduction to an Open Source Geostack.* http://workshops.opengeo.org/stack-intro/openlayers.html#a-map-with-a-styled-wfs-layer (Zugriff am 12. 06 2009).

[13] *JavaScript Toolkit for Rich Web Mapping Applications.* http://www.geoext.org/ (Zugriff am 15. 08 2009).

[14] Korduan/Zehner. *Geoinformation im Internet.* Heidelberg, München, Landsberg, Berlin: Wichmann, 2008.

[15] *Learning Ext.* http://myext.blogspot.com/2007/08/test.html (Zugriff am 15. 06 2009).

[16] Microsoft. *Internet Explorer 8: Startseite.* http://www.microsoft.com/germany/windows/internet-explorer/ (Zugriff am 23. 09 2009).

[17] Mitchell, Tyler. *Web Mapping mit Open Source-GIS-Tools.* Köln: O'Reilly Verlag, 2005.

[18] *OpenJUMP – The free, Java-based and open source Geographic Information System for the World.* http://www.openjump.org/wiki/show/HomePage (Zugriff am 22. 05 2009).

[19] *OpenLayers/MapFish demo - WFS-T on BDGA data.* http://dev4.mapgears.com/bdga-mapfish/bdgaWFS-T.html (Zugriff am 01. 08 2009).

[20] *OpenLayers: Free Maps for the Web* . http://openlayers.org/ (Zugriff am 14. 07 2009).

[21] OSGeo. *OSGeo - You Open Source Compass.* 12. Juli 2009. http://www.osgeo.org/mapserver (Zugriff am 12. Juli 2009).

[22] *PostgreSQL das freie objektrelationale Open Source Datenbanksystem.* http://www.postgresql.de/ (Zugriff am 01. 05 2009).

[23] Rinne, Marco. *Komfort-Browser oder Warmduscher?* http://www.chip.de/artikel/Funktions-Check-Firefox-Opera-und-IE-im-Vergleich_16742561.html (Zugriff am 12. 06 2009).

[24] *Webbrowser Firefox, schneller, sicherer, intelligenter, besser.* http://www.mozilla-europe.org/de/firefox/ (Zugriff am 23. 09 2009).

[25] *What is a Store? How does it work?* http://www.quizzpot.com/2009/05/what-is-store-how-does-it-work/ (Zugriff am 15. 06 2009).

[26] *Willkommen beim Quantum GIS Projekt.* http://www.qgis.org/ (Zugriff am 12. 05 2009).

Abbildungsverzeichnis

Beispielverzeichnis

Tabellenverzeichnis

Definitionsverzeichnis

Anhang

1 Quellcode

1.1 index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Immobilien Präsentation</title>
<!--Styles -->
    <link rel="stylesheet" type="text/css" href="client/mfbase/ext/resources/css/ext-all.css" />
<!--eigenes Style für Buttonimages etc.-->
    <link rel="stylesheet" type="text/css" href="style.css" />
<!--Funktionen,vorgabe von Mapfish -->
    <script type="text/JavaScript"
src="client/mfbase/openlayers/lib/Firebug/firebug.js"></script>
    <script type="text/JavaScript" src="client/mfbase/openlayers/lib/OpenLayers.js"></script>

    <script type="text/JavaScript" src="client/mfbase/ext/adapter/ext/ext-base.js"></script>
    <script type="text/JavaScript" src="client/mfbase/ext/ext-all-debug.js"></script>
    <script type="text/JavaScript" src="client/mfbase/geoext/lib/GeoExt.js"></script>
    <script type="text/JavaScript">
      // Because of a bug in Firefox 2 we need to specify the MapFish base path.
      // See https://bugzilla.mozilla.org/show_bug.cgi?id=351282
      var gMfLocation = "client/mfbase/mapfish/";
    </script>

    <script type="text/JavaScript" src="client/mfbase/mapfish/MapFish.js"></script>
 <!--eigene Funktionen -->

    <script type="text/JavaScript" src="jsfunktionen/info.js"></script>
    <script type="text/JavaScript" src="jsfunktionen/mapoptions.js"></script>
    <script type="text/JavaScript" src="jsfunktionen/layertree.js"></script>
    <script type="text/JavaScript" src="jsfunktionen/toolbar.js"></script>
    <script type="text/JavaScript" src="jsfunktionen/wfs_layer.js"></script>
    <script type="text/JavaScript" src="jsfunktionen/wms_layer.js"></script>
    <script type="text/JavaScript" src="jsfunktionen/messen.js"></script>
    <script type="text/JavaScript" src="jsfunktionen/suche.js"></script>

<!-- Koordinatentransformation -->
<script src="client/proj4js/lib/proj4js-compressed.js"></script>
<script src="client/proj4js/lib/defs/EPSG2398.js"></script>

    <script type="text/JavaScript">

        var map, selectControl, selectedFeature,toolbarItems = [], action, actions = {}; // for
debug
        var centered = false;
        //
        OpenLayers.ProxyHost = "/cgi-bin/proxy.cgi?url=";
        printConfigUrl = "http://localhost:8080/print-servlet-1.1/pdf/info.json";
```

```
Ext.onReady(function() {
    //Karte und Optionen laden
    var map = createMap();

    //WMS Layer laden
    createUebersichtLayer(map);
    createOrthophotoLayer(map);
    createInfrastrukturLayers(map);
    createWheregroupLayer(map);
    createLeerenLayer(map);

    //WFS Layer & Storesladen
    gebaeudestore(map);
    flurststore(map);
    alleraeumestore(map);

    //zusätzliche Funktionen laden
    wheelZoom();
    createItems(map);
    //Optik
    var model = createTreeModel();
   // var toolbar = createToolbar(map);
    var layerOverrides={ "OpenLayers WMS": {overview: true},Countries: { format:
'image/svg+xml' }};
    viewport = new Ext.Viewport({
        layout: "border",
        listeners: {
            afterlayout: centerMap,
            scope: map
        },
        items: [{
            region: "west",
            id: "tools",
            title: "Werkzeuge",
            border: true,
            width: 250,
            autoheight:true,
            split: true,
            collapsible: true,
            tbar: toolbarItems,
            items: [{border:true,
                    height: (window.innerHeight/100)*60-55,
                    layout:"accordion",
                    items:[{
                        id: "tree",
                        title: "Layer",
                        xtype: "layertree",
                        map: map,
                        autoScroll:true,
                        layout:"fit",
                        showWmsLegend: true,
```

```
                        enableDD: true,
                        model: model,
                        Plugins: [

mapfish.widgets.LayerTree.createContextualMenuPlugin(['remove','opacitySlideDirect','zoom
ToExtent'])
                       ]
                       },{
                        id: "search",
                        title: "Suche",
                        map: map,
                        html: '<select id="wechsel" onChange="suchwechsel()"><option
selected>Auswahl</option><option>Flurstücke</option><option>Gebäude</option><option
>Räume</option></select><div id="auswahl"></div>',
                        autoHeight:true,
                        layout:"fit",
                        enableDD: true,
                        autoScroll:true
                       },{
                        xtype: 'print-simple',
                        title: 'Drucken',
                        layout:"fit",
                        autoHeight: true,
                        bodyStyle: 'padding: 7px;',
                        formConfig: {
                                labelWidth: 65,
                                defaults: {
                                        width: 140,
                                        listWidth: 140
                                      },
                                items: [{
                                      xtype: 'textfield',
                                      fieldLabel:
OpenLayers.Lang.translate('mf.print.mapTitle'),
                                      name: 'mapTitle',
                                      value: 'Map title'
                                      },{
                                      xtype: 'textarea',
                                      fieldLabel:
OpenLayers.Lang.translate('mf.print.comment'),
                                      name: 'comment',
                                      height: 100,
                                      value: 'Some comments'
                                      }
                                      ]},
                        border: false,
                        map: map,
                        configUrl: printConfigUrl,
                        overrides: layerOverrides
                    }]}]
            }, {
```

```
            region: "center",

            id: "map",
            title: "Gelände RFH",
            layout: "fit",
            split: true,
            xtype: 'mapcomponent',
            map: map
        }, {
            region: "south",
            id:'tabpanels',
            title: 'Tabpanels',
            xtype: 'tabpanel',
            activeTab:2,
            split: false,
            collapsible: true,
            autoScroll:true,
            height: (window.innerHeight/100)*40,
            items:[{
                title: 'Flurstücke',
                id:'flurstueck',
                        layout: 'fit',
                        items:[rfhflurPanel]
                },{
                title: 'Gebäude',
                        layout: 'fit',
                        id: 'gebaeude',
                        items:[gebaeudePanel]
                },{
                title: 'alle Räume',
                        layout: 'fit',
                        id:'raeume',
                        items:[alleraeumePanel]
                }]
        },{
            region: "east",
            id: "east",
            title: "Information",
            split: true,
            autoheight:true,
            autoScroll:true,
            width:350,
            collapsed: true,
            collapsible: true,
            html: '<div id="information"></div><div style="visibility:hidden"
id="status">1</div>'
        }]
    });
    viewport.doLayout();
    /*rfhflurPanel.getSelectionModel().on('rowselect', function() {
    Ext.getCmp('east').collapse();});
```

```
        gebaeudePanel.getSelectionModel().on('rowselect', function() {
        Ext.getCmp('east').collapse();});
        alleraeumePanel.getSelectionModel().on('rowselect', function() {
        Ext.getCmp('east').collapse();});*/
      });


    --></script>

  </head>

  <body>
    <div id="load"></div>
    <div id="map"></div>
  </body>
</html>
```

1.2 info.js

```
//Select - Information Funktionen
//Gebaeude
function onFeatureSelectGebaeude(feature) {
selectedFeature = feature;
Ext.getCmp('east').expand();
document.getElementById('information').innerHTML='<div
id="gebaeudebild"></div>ID:<div id="gebaeudeid"></div>Nummer:<div
id="gebaeudenummer"></div>Bezeichnung<div id="gebaeudebezeichnung"></div>';
document.getElementById('gebaeudebild').innerHTML='<img
src="images/'+feature.data.bild+'" width="100" height="100" border="0" alt="Gebaeude">';
document.getElementById('gebaeudeid').innerHTML=feature.data.gebaeudeid;
document.getElementById('gebaeudenummer').innerHTML=feature.data.gebaeudenummer;
document.getElementById('gebaeudebezeichnung').innerHTML=feature.data.bezeichnung;
document.getElementById('status').innerHTML='1';}
//Raeume
function onFeatureSelectRaeume(feature) {
selectedFeature = feature;
Ext.getCmp('east').expand();
document.getElementById('information').innerHTML='<div
id="gebaeudebild"></div>ID:<div id="gebaeudeid"></div>Nummer:<div
id="gebaeudenummer"></div>Bezeichnung<div id="gebaeudebezeichnung"></div>';
document.getElementById('gebaeudebild').innerHTML='<img
src="images/'+feature.data.bild+'" width="100" height="100" border="0" alt="Raumbild"';
document.getElementById('gebaeudeid').innerHTML=feature.data.gebaeudeid;
document.getElementById('gebaeudenummer').innerHTML=feature.data.gebaeudenummer;
document.getElementById('gebaeudebezeichnung').innerHTML=feature.data.bezeichnung;
document.getElementById('status').innerHTML='1';}
//Flurstuecke
function onFeatureSelectFlur(feature) {
selectedFeature = feature;
Ext.getCmp('east').expand();
```

```
document.getElementById('information').innerHTML='<div
id="gebaeudebild"></div>ID:<div id="gebaeudeid"></div>Nummer:<div
id="gebaeudenummer"></div>Bezeichnung<div id="gebaeudebezeichnung"></div>';
document.getElementById('gebaeudebild').innerHTML='<img
src="images/'+feature.data.bild+'" width="100" height="100" border="0" alt="Flurstück"';
document.getElementById('gebaeudeid').innerHTML=feature.data.gebaeudeid;
document.getElementById('gebaeudenummer').innerHTML=feature.data.gebaeudenummer;
document.getElementById('gebaeudebezeichnung').innerHTML=feature.data.bezeichnung;
document.getElementById('status').innerHTML='1';}
```

1.3 layertree.js

```
function createTreeModel() {
        return [{
          text: "Hintergrund",
          expanded: false,
          children: [{
                  text: "Übersichtskarte",
                  checked: false,
                  layerName:"uebersicht"
           },{
                  text: "Luftbild",
                  checked: true,
                  layerName:"orthophoto"
           },{
                  text: "Wheregroup",
                  checked: false,
                  layerName:"wheregroup"
           },{
                  text: "weißer Hintergrund",
                  checked: false,
                  layerName:"leer"
           }]
        },{text: "Infrastruktur",
          expanded: false,
          checked: false,
          children:[{
                        text: "Trinkwasser",
                        checked: false,
                        layerName: "trinkwasser"
                 },{
                        text: "Regenwasser",
                        checked: false,
                        layerName: "regenwasser"
                 },{
                        text: "Schmutzwasser",
                        checked: false,
                        layerName: "schmutzwasser"
                 },{
                        text: "Fernwärme",
                        checked: false,
```

```
                    layerName: "fernwaerme"
                },{
                    text: "Beleuchtung",
                    checked: false,
                    layerName: "beleuchtung"
                },{
                    text: "Strassen",
                    checked: false,
                    layerName: "strassen"
                },{
                    text: "Telefon",
                    checked: false,
                    layerName: "telefon"
                }]
        },{
        text: "Gebäude",
        checked: false,
        layerName: "gebaeude"
        },{
        text: "Flurstücke",
        checked: false,
        layerName: "flurst"
        },{ text: "Räume",
        checked: false,
        layerName: "alleraeume"
        }];
    };
```

1.4 mapoptions.js

```
function createMap() {
map = new OpenLayers.Map("map",{projection:
"EPSG:2398",controls:[],scales:[442943842.5,221471921.25,110735960.625,55367980.3125,
27683990.15625,13841995.078125,6920997.5390625, 3460498.76953125,
1730249.384765625,865124.6923828125, 432562.34619140625, 216281.17309570312,
108140.58654785156,
            54070.29327392578, 10000.0, 5000.0,1000.0],maxExtent: new
OpenLayers.Bounds(4504884.5,5997999.5,4506172.5,5999757.5)});
        return map;}
  function centerMap() {
        if (!centered) {
            this.setCenter(new
OpenLayers.LonLat(4504884.5,5997999.5,4506172.5,5999757.5));
            centered = true;}}
function wheelZoom(){
        // navigation control
        var navControl = new OpenLayers.Control.Navigation({
            type: OpenLayers.Control.TYPE_TOGGLE,
            zoomWheelEnabled: true});
        map.addControl(navControl);
        navControl.activate();}
```

1.5 messen.js

```
//Zubehör für Strecken und Flächenbemessung
        var measureOptions = {
        persist:true,
        geodesic:true,
                                                handlerOptions : {
                                                  style :"default"
                                                }
                                        }
                                        var lineMeasure = new OpenLayers.Control.Measure(
                                                        OpenLayers.Handler.Path, measureOptions);
                                        var polygonMeasure = new OpenLayers.Control.Measure(
                                                        OpenLayers.Handler.Polygon, measureOptions);
                                        lineMeasure.events.on( {
                                                "measure" :handleDistanceMeasurements,
                                                "measurepartial" :handleDistanceMeasurements
                                        });
                                        polygonMeasure.events.on( {
                                                "measure" :handleAreaMeasurements,
                                                "measurepartial" :handleAreaMeasurements
                                        });
                                        function handleAreaMeasurements(event) {
                                                var units = event.units;
                                                var measure = event.measure;
                                                var element = document.getElementById('information');
                                                var out = "";
                                                out += "Fläche: " + measure.toFixed(3) + " " + units
                                                                + "<sup>2</sup>";
                                                element.innerHTML = out;
                                        }
                                        function handleDistanceMeasurements(event) {
                                                Ext.getCmp('east').expand();
        var units = event.units;
                                                var measure = event.measure;
                                                if
(document.getElementById('status').childNodes[0].nodeValue!='2')
                                                {

          document.getElementById('information').innerHTML='<div
id="teilmessung"></div><br><div id="messung"></div><br><div id="status2"
style="visibility:hidden">käse</div>';
                                                document.getElementById('status').innerHTML='2';
                                                }
                                                else
                                                {};
                                                var teilstreckenanzahl =
document.getElementById('teilmessung').childNodes.length;
                                                var meldung="";
                                                var rechnung=0;
                                                var ergebniss=0;
```

```
                                        var test=document.getElementById('status2');
        var testout="";
        if(document.getElementById('status2').childNodes[0].nodeValue=="2")
        {
        document.getElementById('teilmessung').innerHTML="";
        }
                                        for (var
i=0;i<document.getElementById('teilmessung').childNodes.length;i++)
                                        {

if(document.getElementById('teilmessung').childNodes[i].nodeValue!=null )
                                            {
rechnung=rechnung+Number(document.getElementById('teilmessung').childNodes[i].nodeV
alue);
meldung +=document.getElementById('teilmessung').childNodes[i].nodeValue+"<br>";
            }}
        if (measure.toFixed(3)>0)
        {
        ergebniss=Number(measure.toFixed(3))-rechnung;
        var teilelement = document.getElementById('teilmessung');
        var teilausgabe ="";
        //letzte Strecke mit 0 nicht anzeigen
        if(ergebniss>0){
        teilout =meldung+ergebniss;
        testout="1";
        }
        else
        {
        teilout=meldung;
        testout="2";
        }
        teilelement.innerHTML =teilout;
        test.innerHTML=testout;
                                        // Test: alert(meldung+" "+teilstreckenanzahl+"
"+ergebniss);
                                        }
                                        //Gesamtstrecke
                                        var element = document.getElementById('messung');
                                        var out = "";
                                        out += "Distanz: " + measure.toFixed(3) + " " + units;
                                        element.innerHTML = out;};
```

1.6 suche.js

```
// JavaScript Document
//Suche Filter
//alt beispiel <button type="button" value="Überraschung"
onclick="gebaeudeStore.reload({callback: function(){gebaeudeStore.filter(\'bezeichnung2\',
\'Zoll\');}});"></button>
function suchwechsel() {
  if(document.getElementById("wechsel").value=='Flurstücke')
```

```
  {
  Ext.getCmp('tabpanels').setActiveTab('flurstueck');
  document.getElementById('auswahl').innerHTML='<hr>Flurnummer:<input type="text"
name="nummer" id="id" value="Eingabe"size="10"><p><input type="submit"
onclick="suche()" value="suchen"><input type="submit" onclick="reset()" value="reset">';
  }
  else
  {
  if(document.getElementById("wechsel").value=='Gebäude')
        {
        Ext.getCmp('tabpanels').setActiveTab('gebaeude');
        document.getElementById('auswahl').innerHTML='<hr>GebäudeID:<input
type="text"id="id" name="gebaeudeid" value="Eingabe"size="10"><p><input type="submit"
onclick="suche()" value="suchen"><input type="submit" onclick="reset()" value="reset">';
        }
        else
        {
         if(document.getElementById("wechsel").value=='Räume')
         {
         Ext.getCmp('tabpanels').setActiveTab('raeume');
         document.getElementById('auswahl').innerHTML='<hr>Raumnummer:<input
type="text" id="1" name="raumid" value="Eingabe"size="10"><p>Gebäude:<input
type="text" id="2" name="gebaeudeid" value="Eingabe"size="10"><p>Etage:<input
type="text" id="3" name="etagennr" value="Eingabe"size="10"><p>Mietvertrag:<input
type="text" id="4" name="mietvertra" value="Nummer"size="10"><p><input type="submit"
onclick="suche()" value="suchen"><input type="submit" onclick="reset()" value="reset">';
         }
         else
         {
         document.getElementById('auswahl').innerHTML='';
         }
        }
  }
}
//Filter
function filterid(property,value)
{
 var filter =   new OpenLayers.Filter.Comparison(
              {
               type: OpenLayers.Filter.Comparison.EQUAL_TO,
               property: property,
               value: value
              });
              return filter;
}
function einlesenproperty(anzahl)
{
var i = 1;
var filtername = new Array();
while (i<=anzahl){
if (document.getElementById(i).value!='Eingabe' )
```

```
{
var name = document.getElementById(i).name;
filtername.push(name);
}
i++;
}
return filtername;
}

function einlesenvalue(anzahl)
{
var i = 1;
var filtervalue = new Array();
while (i<=anzahl){
if (document.getElementById(i).value!='Eingabe' )
{
var value = document.getElementById(i).value;
filtervalue.push(value);
}
i++;
}
return filtervalue;
}
function auswertung (property,value)
{
var i = property.length;
var a=1;
while(a<=i)
{
return filterid(property[a-1],value[a-1]);
a++;
}
}
function suche(){
  var art=document.getElementById('wechsel').value;
  if( art=="Räume")
  {
  var anzahl = 4;
  var filterproperty=einlesenproperty(anzahl);
  var filtervalue=einlesenvalue(anzahl);
  if(auswertung(filterproperty,filtervalue)!=null){
alleraeumeStore.load(alleraeumeStore.proxy.protocol.filter.filters=[auswertung(filterproperty,
filtervalue)]);
  alert(raeume.getDataExtent());
  map.zoomToExtent(raeume.getDataExtent());
  }
  else{}
  }
  else
  {
    if( art=="Gebäude")
```

```
    {
    var property ='gebaeudeid';
    var value =document.getElementById('id').value;
    gebaeudeStore.load(gebaeudeStore.proxy.protocol.filter.filters=[filterid(property,value)]);
    map.zoomToExtent(gebaeude.getDataExtent());
    }
    else
    {
    var property ='objectid';
    var value =document.getElementById('id').value;
    flurstStore.load(flurstStore.proxy.protocol.filter.filters=[filterid(property,value)]);
    map.zoomToExtent(flurst.getDataExtent());
    }
  }
  }

function reset(){
  var art=document.getElementById('wechsel').value;
  if( art=="Räume")
  {
  alleraeumeStore.load(alleraeumeStore.proxy.protocol.filter.filters=[]);
  map.zoomToExtent(raeume.getDataExtent());
  }
  else
  {
    if( art=="Gebäude")
    {
    gebaeudeStore.load(gebaeudeStore.proxy.protocol.filter.filters=[]);
    map.zoomToExtent(gebaeude.getDataExtent());
    }
    else
    {
    flurstStore.load(flurstStore.proxy.protocol.filter.filters=[]);
    map.zoomToExtent(flur.getDataExtent());}}}
```

1.7 toolbar.js

```
// JavaScript Document
function createItems(map) {
var action;
var createSeparator = function() {
            toolbarItems.push(" ");
            toolbarItems.push("-");
            toolbarItems.push(" ");
          };
action = new GeoExt.Action({
             control: new OpenLayers.Control.ZoomToMaxExtent(),
             map: map,
             iconCls: 'zfull',
             tooltip: 'Zoom to full extent'
          });
```

```
            toolbarItems.push(action);
            createSeparator();
action = new GeoExt.Action({
                control: new OpenLayers.Control.ZoomBox(),
                tooltip: 'Zoom in: click in the map or use the left mouse button and drag to create a
rectangle',
                map: map,
                iconCls: 'zoom',
                toggleGroup: 'map'
            });
            toolbarItems.push(action);
action = new GeoExt.Action({
                control: new OpenLayers.Control.ZoomBox({
                    out: true
                }),
                tooltip: 'Reinzoomen, einfach Klick in die Karte oder Rechteck aufziehen',
                map: map,
                iconCls: 'zout',
                toggleGroup: 'map'
            });
            toolbarItems.push(action);
            createSeparator();


action = new GeoExt.Action({
            control: lineMeasure,
            tooltip: 'Distanzmessung',
            map:map,
            iconCls: 'line',
            toggleGroup: 'map',
            })  ;
            toolbarItems.push(action);

action = new GeoExt.Action({
            control: polygonMeasure,
            map:map,
            tooltip: 'Flächenmessung',
            iconCls: 'polygon',
            toggleGroup: 'map'
            }) ;
            toolbarItems.push(action);
            createSeparator();
action = new GeoExt.Action({
          iconCls: 'info',
          tooltip: 'Gebäudeinformationen',
          control: new OpenLayers.Control.SelectFeature(gebaeude, {
          type: OpenLayers.Control.TYPE_TOGGLE,
          onSelect: onFeatureSelectGebaeude
        }),
          map: map,
          toggleGroup: 'map',
```

```
        group:'info'
       });
        toolbarItems.push(action);
action = new GeoExt.Action({
        iconCls: 'info',
        tooltip: 'Rauminformationen',
        control: new OpenLayers.Control.SelectFeature(raeume, {
        type: OpenLayers.Control.TYPE_TOGGLE,
        onSelect: onFeatureSelectRaeume}),
        map: map,
        toggleGroup: 'map',
        group:'info'});
        toolbarItems.push(action);
action = new GeoExt.Action({
        iconCls: 'info',
        tooltip: 'Informationen zu den Flurstücken',
        control: new OpenLayers.Control.SelectFeature(flurst, {
        type: OpenLayers.Control.TYPE_TOGGLE,
        onSelect: onFeatureSelectFlur       }),
        map: map,
        toggleGroup: 'map',
        group:'info'    });
   toolbarItems.push(action);}
```

1.8 wfs_layer.js

```
var defStyleFlur = new OpenLayers.Style({
  fillColor: "green",
  fillOpacity:0.7,
  strokeColor: "green",
  strokeWidth:1.0
});
var selStyleFlur = new OpenLayers.Style({
  fillColor: "red",
  fillOpacity:1.0,
  strokeColor: "red"
});
var flurStyle = new OpenLayers.StyleMap({
   "default": defStyleFlur,
   "select": selStyleFlur
});
var defStyleRaeume = new OpenLayers.Style({
  fillColor: "#ffffff",
  fillOpacity:0.7,
  strokeColor: "#000000",
  strokeWidth:1.0
});
var selStyleRaeume = new OpenLayers.Style({
  fillColor: "red",
  fillOpacity:1.0,
  strokeColor: "red"
```

```
});
var raeumeStyle = new OpenLayers.StyleMap({
   "default": defStyleRaeume,
   "select": selStyleRaeume
});
var defStyleGebaeude = new OpenLayers.Style({
  fillColor: "#ffffff",
  fillOpacity:0.7,
  strokeColor: "#000000",
  strokeWidth:1.0
});
var selStyleGebaeude = new OpenLayers.Style({
  fillColor: "red",
  fillOpacity:1.0,
  strokeColor: "red"
});
var gebaeudeStyle = new OpenLayers.StyleMap({
   "default": defStyleGebaeude,
   "select": selStyleGebaeude
});
//WFS Layer alle Gebäude
function gebaeudestore(map){
gebaeude = new OpenLayers.Layer.Vector("gebaeude",{styleMap: gebaeudeStyle});
map.addLayer(gebaeude);

gebaeudeStore = new GeoExt.data.FeatureStore({
            layer: gebaeude,
            autoDestroy:true,
            proxy: new GeoExt.data.ProtocolProxy({
            protocol: new OpenLayers.Protocol.WFS({
                  url:  "http://localhost/geoserver/wfs",
                  featureType: "rfhgebaeude",
                  featureNS: "http://www.rostock.de/",
                  srsName: "EPSG:2398",
                  version: "1.1.0",
                  extractAttributes: true,
                  filter: new OpenLayers.Filter.Logical({
                        type:OpenLayers.Filter.Logical.AND,
                        filters:[]
                        })
                  })
            }),
            fields: [
            {name: 'gebaeudeid', type: 'string'},
            {name: 'gebaeudenummer', type: 'string'},
            {name: 'bezeichnung', type: 'string'},
            {name: 'bezeichnung2', type: 'string'},
            {name: 'bild', type: 'string'},
            ],
            autoLoad: true
            });
```

```
gebaeudePanel = new Ext.grid.GridPanel({
                region: "south",
                store: gebaeudeStore,
                width: 320,
                loadMask : {msg: 'Bitte Warten...'},
                autoheight: true,
                columns: [{
                        header: "GebäudeID",
                        width: 100,
                        sortable: true,
                        dataIndex: "gebaeudeid"
                       }, {
                        header: "Gebäudenummer",
                        width: 100,
                        sortable: true,
                        dataIndex: "gebaeudenummer"
                       }, {
                        header: "Bezeichnung",
                        width: 100,
                        sortable: true,
                        dataIndex: "bezeichnung"
                       }, {
                        header: "Sonstiges",
                        width: 100,
                        sortable: true,
                        dataIndex: "bezeichnung2"
                       }, {
                        header: "Bild",
                        width: 100,
                        sortable: true,
                        dataIndex: "bild"
                       }],
                sm: new GeoExt.grid.FeatureSelectionModel()
                });
}

function alleraeumestore(map){
raeume = new OpenLayers.Layer.Vector("alleraeume");
        map.addLayers([raeume]);

alleraeumeStore = new GeoExt.data.FeatureStore({
            layer: raeume,
            proxy: new GeoExt.data.ProtocolProxy({
            protocol: new OpenLayers.Protocol.WFS({
                    url:  "http://localhost/geoserver/wfs",
                    featureType: "rfhalleraeume",
                    featureNS: "http://www.rostock.de/",
                    srsName: "EPSG:2398",
                    version: "1.1.0",
                    extractAttributes: true,
```

```
                filter: new OpenLayers.Filter.Logical({
                        type:OpenLayers.Filter.Logical.AND,
                        filters:[]
                        })
                })
        }),
        fields: [
        {name: 'raumid', type: 'string'},
        {name: 'raumnr', type: 'string'},
        {name: 'gebaeudeid', type: 'string'},
        {name: 'etagennr', type: 'int'},
        {name: 'nutzung', type: 'string'},
        {name: 'bemerkung', type: 'string'},
        {name: 'nichtnutzb', type: 'float'},
        {name: 'nebenm', type: 'float'},
        {name: 'groesse', type: 'float'},
        {name: 'mietende', type: 'date'},
        {name: 'mietbeginn', type: 'date'},
        {name: 'mietvertra', type: 'string'},
        {name: 'kaltmiete', type: 'float'},
        {name: 'betriebsko', type: 'float'},
        {name: 'elektro', type: 'float'},
        {name: 'heizung', type: 'float'},
        {name: 'wasser', type: 'float'},
        {name: 'kaution', type: 'float'}
        ],
        autoLoad: true
        });
alleraeumePanel = new Ext.grid.GridPanel({
                region: "south",
                store: alleraeumeStore,
                width: 320,
                loadMask : {msg: 'Bitte Warten...'},
                autoheight: true,
                columns: [{
                        header: "RaumID",
                        width: 60,
                        dataIndex: "raumid"
                        }, {
                        header: "RaumNr.",
                        width: 60,
                        dataIndex: "raumnr"
                        }, {
                        header: "EtageNr.",
                        width: 25,
                        dataIndex: "etagennr"
                        }, {
                        header: "GebäudeID",
                        width: 60,
                        dataIndex: "gebaeudeid"
                        }, {
```

```
  header: "Nutzung",
  width: 60,
  dataIndex: "nutzung"
}, {
  header: "Anmerkung",
  width: 100,
  dataIndex: "bemerkung"
}, {
  header: "Größe",
  width: 100,
  dataIndex: "groesse"
}, {
  header: "Unnutzbar",
  width: 100,
  dataIndex: "nichtnutzb"
}, {
  header: "NebenQM",
  width: 100,
  dataIndex: "nebenm"
}, {
  header: "Mietvertrag",
  width: 100,
  dataIndex: "mietvertra"
}, {
  header: "Mietbeginn",
  width: 100,
  dataIndex: "mietbeginn"
}, {
  header: "Mietende",
  width: 100,
  dataIndex: "mietende"
}, {
  header: "Kaltmiete",
  width: 100,
  dataIndex: "kaltmiete"
}, {
  header: "Btr.Ko.",
  width: 100,
  dataIndex: "betriebsko"
}, {
  header: "Elek.",
  width: 100,
  dataIndex: "elektro"
}, {
  header: "Heizung",
  width: 100,
  dataIndex: "heizung"
}, {
  header: "Wasser",
  width: 100,
  dataIndex: "wasser"
```

```
                        }, {
                          header: "Kaution",
                          width: 100,
                          dataIndex: "kaution"
                        }],
                    sm: new GeoExt.grid.FeatureSelectionModel()
                    });
}

function flurststore(map){
flurst = new OpenLayers.Layer.Vector("flurst",{styleMap: flurStyle});
        map.addLayer(flurst);

flurstStore = new GeoExt.data.FeatureStore({
            layer: flurst,
            proxy: new GeoExt.data.ProtocolProxy({
            protocol: new OpenLayers.Protocol.WFS({
                    url:  "http://localhost/geoserver/wfs",
                    featureType: "rfhflurstu",
                    featureNS: "http://www.rostock.de/",
                    srsName: "EPSG:2398",
                    version: "1.1.0",
                    extractAttributes: true,
                     filter: new OpenLayers.Filter.Logical({
                            type:OpenLayers.Filter.Logical.AND,
                            filters:[]
                            })
                    })
            }),
            fields: [
            {name: 'objectid', type: 'int'},
            {name: 'zaehler', type: 'int'},
            {name: 'nenner', type: 'string'},
            {name: 'z_n', type: 'string'},
            {name: 'bem', type: 'string'},
            {name: 'status_nut', type: 'string'}
            ],
            autoLoad: true
            });
rfhflurPanel = new Ext.grid.GridPanel({
                    region: "south",
                    store: flurstStore,
                    width: 320,
                    loadMask : true,
                    autoheight: true,
                    columns: [{
                            header: "FlurID",
                            width: 100,
                            dataIndex: "objectid"
                        },{
                            header: "Zähler",
```

```
                    width: 100,
                    dataIndex: "zaehler"
                  },{
                    header: "Nenner",
                    width: 100,
                    dataIndex: "nenner"
                  },{
                    header: "Z/N",
                    width: 100,
                    dataIndex: "z_n"
                  },{
                    header: "Bemerkung",
                    width: 100,
                    dataIndex: "bem"
                  },{
                    header: "Status",
                    width: 100,
                    dataIndex: "status_nut"
                  }
                  ],
            sm: new GeoExt.grid.FeatureSelectionModel()
            });
}
```

1.9 wms_layer.js

```
function createUebersichtLayer(map) {
        uebersicht =  new OpenLayers.Layer.WMS(
           "uebersicht",
           "http://www.geodaten-mv.de/dienste/gdimv_dtk",
           {layers: "gdimv_dtk"},
           {isBaseLayer: true}
        );
        map.addLayer(uebersicht);
    }
    function createOrthophotoLayer(map) {
        orthophoto = new OpenLayers.Layer.WMS("orthophoto",
           "http://www.geodaten-mv.de/dienste/adv_dop",
            {layers: "adv_dop"},
            {singleTile: true}
            );
        map.addLayer(orthophoto);
    }
    function createWheregroupLayer(map) {
        wheregroup = new OpenLayers.Layer.WMS("wheregroup",
           "http://osm.wheregroup.com/cgi-
bin/mapserv?map=/data/umn/osm/osm_basic.map",
            {layers: "OSM_Basic"},
            {singleTile: true}
            );
        map.addLayer(wheregroup);
```

```
    }
    function createLeerenLayer(map) {
        var leer = new OpenLayers.Layer.WMS("leer",
            "", {isBaseLayer:false}
            );
        map.addLayer(leer);
    }
function createInfrastrukturLayers(map) {
        var trinkwasser = new OpenLayers.Layer.WMS("trinkwasser",
            "http://localhost:8080/geoserver/wms?service=wms", {
                layers: "inftrinkwasser",
                format: "image/png",
                transparent: "true"
            }, {
                singleTile: true
        });
        var regenwasser = new OpenLayers.Layer.WMS("regenwasser",
            "http://localhost:8080/geoserver/wms?service=wms", {
                layers: "infregenwasser",
                format: "image/png",
                transparent: "true"
            }, {
                singleTile: true
        });
        var schmutzwasser = new OpenLayers.Layer.WMS("schmutzwasser",
            "http://localhost:8080/geoserver/wms?service=wms", {
                layers: "infschmutzwasser",
                format: "image/png",
                transparent: "true"
            }, {
                singleTile: true
        });
        var fernwaerme = new OpenLayers.Layer.WMS("fernwaerme",
            "http://localhost:8080/geoserver/wms?service=wms", {
                layers: "inffernwaerme",
                format: "image/png",
                transparent: "true"
            }, {
                singleTile: true
        });
        var strassen = new OpenLayers.Layer.WMS("strassen",
            "http://localhost:8080/geoserver/wms?service=wms", {
                layers: "infstrassen",
                format: "image/png",
                transparent: "true"
            }, {
                singleTile: true
        });
        var telefon = new OpenLayers.Layer.WMS("telefon",
            "http://localhost:8080/geoserver/wms?service=wms", {
                layers: "inftelefon",
```

```
            format: "image/png",
            transparent: "true"
        }, {
            singleTile: true
    });
     var beleuchtung = new OpenLayers.Layer.WMS("beleuchtung",
        "http://localhost:8080/geoserver/wms?service=wms", {
            layers: "infbeleuchtung",
            format: "image/png",
            transparent: "true"
        }, {
            singleTile: true
    });

map.addLayers([beleuchtung,telefon,strassen,fernwaerme,schmutzwasser,regenwasser,trinkw
asser]);
    }
```

1.10 style.css

```
    BODY
    {
    color:#000000;
    font-family:tahoma,arial,helvetica;
    font-size:11px;
    font-size-adjust:true;
    font-stretch:normal;
    font-style:normal;
    font-variant:normal;
    font-weight:normal;
    line-height:normal;
    }
    select {
color:#000000;
    font-family:tahoma,arial,helvetica;
    font-size:11px;
    font-size-adjust:true;
    font-stretch:normal;
    font-style:normal;
    font-variant:normal;
    font-weight:normal;
    line-height:normal;
}
button {
color:#000000;
    font-family:tahoma,arial,helvetica;
    font-size:11px;
    font-size-adjust:true;
    font-stretch:normal;
    font-style:normal;
    font-variant:normal;
```

```
        font-weight:normal;
        line-height:normal;
}
input {
color:#000000;
        font-family:tahoma,arial,helvetica;
        font-size:11px;
        font-size-adjust:true;
        font-stretch:normal;
        font-style:normal;
        font-variant:normal;
        font-weight:normal;
        line-height:normal;
}
        .zoom {
          background-image:url(icons/zoom.png)!important ;
          height:20px !important;
          width:20px !important;
        }
        .zout {
          background-image:url(icons/zout.png)!important ;
          height:20px !important;
          width:20px !important;
        }
        .zfull {
          background-image:url(icons/zfull.png) !important;
          height:20px !important;
          width:20px !important;
        }
        .pan {
          background-image:url(icons/pan.png)!important;
          height:20px !important;
          width:20px !important;
        }
        .info {
            background-image:url(icons/info.png)!important;
            height:20px !important;
            width:20px !important;
        }
        .line {
            background-image:url(icons/line.png) !important;
            height:20px !important;
            width:20px !important;
        }
        .polygon {
            background-image:url(icons/area.png) !important;
            height:20px !important;
            width:20px !important;
        }
        .zooma {
          background-image:url(icons/zoom.png)!important ;
```

```
  height:20px !important;
  width:20px !important;
}
.x-grid3-hd-row td.ux-filtered-column {
font-style: italic;
font-weight: bold;
}

.x-grid3-row-body p {
margin:5px 5px 10px 5px !important;
}
```

2 CD-Inhalt

- Webanwendung mit den selbstgeschrieben Funktionen und den Mapfish-Bibliotheken
- PDF-Dokument der Masterarbeit

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und nur die angegebenen Quellen und Hilfsmittel genutzt habe.

Rostock, 12.10.2009