Routenplaner für einen Emergency Route Service auf Basis der OpenLS^{TM} Spezifikation

Diplomarbeit

Fachhochschule Mainz Fachbereich I
Studiengang Geoinformatik und Vermessung

Pascal Neis
(MatrikelNr.: 505007)

Betreuer : Prof. Dr. Alexander Zipf

Bearbeitungszeitraum: 15.02.2006 - 15.08.2006

Danksagung

Mein erster Dank geht an alle, die zum Gelingen dieser Diplomarbeit beigetragen haben.

Herrn Prof. Dr. Alexander Zipf möchte ich für das interessante Thema und die gute Betreuung während der Durchführung dieser Diplomarbeit danken.

Mein spezieller Dank gilt meiner Familie und meiner Freundin, ohne deren Unterstützung und Geduld ich nicht so weit gekommen wäre.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADT	Abstract Data Type
AOI	Area of Interest
API	Application Programming Interface
DB	Datenbank
DTD	Document Type Definition
EPSG	European Petroleum Survey Group
ERS	Emergency Route Service
ESRI	Environmental Systems Research Institute
Frida	Freie Vektor-Geodaten Osnabrück
GIS	Geoinformationssystem
GML	Geography Markup Language
GMS	GeoMobility Server
GPL	General Public License
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
JAR	Java Archive
JDBC	Java Database Connectivity
JDK	Java Development Kit
JPEG	Joint Photographic Experts Group
JRE	Java Runtime Environment
JTS	Java Topology Suite
JVM	Java Virtual Machine
LBS	Location Based Services
LGPL	Lesser General Public License
MLP	Mobile Location Protocol
OGC	Open Geospatial Consortium
OpenLS	Open Location Services

PNG	Portable Network Graphics
POI	Point of Interest
RS	Route Service
SLD	Styled Layer Descriptor
SRS	Spatial Reference System
SVG	Scalable Vector Graphics
UML	Unified Modeling Language
UMN	University of Minnesota
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WFS	Web Feature Service
WFS-T	Web Feature Service Transactional
WMS	Web Map Service
WWW	World Wide Web
XLS	XML for Location Services
XML	Extensible Markup Language
XSD	XML-Schemata

1 Einleitung

Damit Einsatzkräfte möglichst schnell zu ihrem Einsatzort kommen, ist ein Routenplaner von sehr hoher Bedeutung. Nur was tun, wenn die Fahranweisungen die Lebensretter geradewegs in ein nicht passierbares Gebiet oder auf eine nicht befahrbare Straße führen?

Der Titel dieser Diplomarbeit lautet:

„Routenplaner für einen Emergency Route Service auf Basis der OpenLS^{TM} Spezifikation“

Zunächst möchte ich den Titel meiner Diplomarbeit in eine etwas verständlichere Form bringen, um einen Zusammenhang der einzelnen Begriffe mit dem erwähnten Anwendungsfall herzustellen. Viele Menschen kennen mittlerweile gewöhnliche *Routenplaner*[1] durch Navigationssysteme im Auto oder PDA[2]. Mit der *OpenLS Spezifikation* können wahrscheinlich die Wenigsten etwas anfangen, geschweige denn mit einem *Emergency Route Service*. Letzterer Begriff kann wie folgt kurz erklärt werden indem der Unterschied zu bereits Bekanntem aufgezeigt wird:

- *Route Service*
  Vergleichbar mit einem Routenplaner wie es sie zahlreich im Internet gibt.
- *Emergency Route Service*
  Entspricht einem normalen Route Service, aber mit der Besonderheit, dass bei der Ermittlung der Route aktuelle Gefahrengebiete oder Straßensperrungen berücksichtigt und umfahren werden können.

Zurück zum Thema: Die OpenLS Spezifikation ist ein Dokument, in welchem ein Route Service standardisiert ist, der es ermöglicht eine Grundlage bereitzustellen, die das anfangs genannte Problem lösen könnte. Standardisiert bedeutet, dass für den in der Spezifikation beschriebenen Web Service[3] die Schnittstellen (wie die Anwendung anzusprechen

[1] aus Wikipedia[34]: Routenplaner (Streckenplaner, Wegplaner, von französisch: route = Weg) sind Computerprogramme, mit deren Hilfe ein Weg zwischen einem Start- und einem Zielort gefunden werden kann.

[2] PDA - Personal Digital Assistant

[3] Software Anwendung die auf einem Server läuft

ist, welche Möglichkeiten es unterstützen sollte und wie die Antworten von ihr auszusehen haben) schon vor dieser Diplomarbeit festgeschrieben waren. Dies bedeutet allerdings, dass diese Spezifikation lediglich eine Art Bauplan ist, um einen solchen Service zu erstellen. Sie sagt nichts darüber aus, wie die eigentliche Anwendung, also der Routenplaner, mit seinen Eigenschaften, wie z.B. Route berechnen, Fahranweisungen oder Karten erstellen, zu entwickeln ist.

1.1 Zielsetzung

Das Ziel der Diplomarbeit habe ich in zwei Teile gegliedert:

- Entwicklung von Java[4]-Klassen[5], mit welchen es möglich ist den eben erwähnten Routenplaner zu erstellen, welcher wiederum Routen unter Berücksichtigung von gesperrten Bereichen planen kann.
- Implementierung der OpenLS Route Service Spezifikation, um die entwickelten Java-Klassen eines Routenplaners als Web Service nutzen zu können.

Bei der Entwicklung der Java-Klassen zur Routenplanung und der Implementierung des Web Services sollen vorhandene Java-Bibliotheken genutzt und ggf. den Anforderungen angepasst werden.

Der Schwerpunkt dieser Diplomarbeit liegt nicht darin, die volle OpenLS Route Service Spezifikation umzusetzen. Dieses wäre aus zeitlichen und umfangreichen Gründen nicht möglich. Vielmehr sollte ein Web Service erstellt werden, der zur Lösung des erwähnten Problems beiträgt, als Grundlage wiederverwendbar ist und in dem verschiedene Anwendungsteile, wie z.B. Routing Algorithmus, der Graph auf dem die Route berechnet wird oder die Erweiterung der Fahranweisungen um Landmarken, austausch- und erweiterbar sind.

[4] Java ist eine Programmiersprache
[5] Eine oder mehrere Klassen bilden ein Programm.

1.2 Motivation

Für was wird ein OpenLS konformer Route Service benötigt?
Mit einem OpenLS Route Service wäre die Grundlage geschaffen, um einen Emergency Route Service zu implementieren. Dieser könnte im Ernstfall Menschenleben retten. Das Anfangs in der Einleitung genannte Problem, dass Einsatzkräfte nicht auf dem schnellsten Weg zum Einsatzort kommen, zeigt die Wichtigkeit eines Emergency Route Service. Es gibt nichts Schlimmeres für einen Menschen der Hilfe benötigt, als auf diese warten zu müssen. Das tägliche Aufkommen von Straßensperrungen durch Unfälle, Demonstrationen oder Sperrungen aus Sicherheitsgründen wegen einer Bombenentschärfung rechtfertigt des Weiteren die Implementierung des OpenLS Route Service als Grundlage für einen Emergency Route Service.

Durch die steigende Anzahl von Naturkatastrophen, könnte der OpenLS Route Service zusätzlich in einem Katastrophen-Management-System genutzt werden und Experten bei der Einschätzung von Notfallsituationen und der Planung von Hilfsmaßnahmen unterstützen (z.B. Evakuierung von Menschen aus Gebieten, welche sich in unmittelbarer Nähe von überschwemmten oder gefährdeten Bereichen befinden).

Ein weiterer Vorteil meiner Diplomarbeit, welcher sich aber erst während der Durchführung herausstellte, ist, dass der zu implementierende OpenLS Route Service mit seinem zu entwickelnden Routenplaner auch zur Prävention genutzt werden könnte. Der Polizei zum Beispiel wäre es möglich, bei Sperrung eines Gebietes oder einer Straße mögliche Umleitungen und Übersichtskarten mit alternativen Routen schon im Vorfeld zu erstellen und die notwendigen Maßnahmen einzuleiten.

Einen Web Service anhand von Spezifikationen und auf Basis von Open-Source[6] Bibliotheken und Anwendungen aufzubauen, war eine weitere Motivation für meine Diplomarbeit.

State of the Art

Gibt es bereits einen Route Service oder Emergency Route Service?
Etwas ähnliches wurde 2003 am *Institut für Geoinformatik* (ifgi) der Westfälischen Wilhelms-Universität Münster im Rahmen eines Studienpojektes („ariadne“ - GI-Dienste für Notfall-Management-Systeme bei Luftschadstoff-Störfällen[7]) erarbeitet. An diesem einjährigen Projekt arbeiteten neun Studenten unter Aufsicht von fünf Betreuern. Es wur-

[6] Open-Source - bedeutet, jeder hat die Erlaubnis in den Programmcode Einblick zu nehmen, ihn zu verändern und weiterzugeben.
[7] Mehr Informationen: http://wwwariadne.uni-muenster.de

den prototypisch verschiedene Dienste (u.a. ein Emergency Route Service und Routing Service) entwickelt, die bei Luftschadstoff-Störfällen durch Chemieunfälle oder zu hohe Verkehrsbelastung, Betroffenen, z.B. Autofahrern, Hilfestellungen leisten. D.h. ein Autofahrer wird während der Fahrt auf mögliche Gefahren hingewiesen und erhält gleichzeitig Informationen, wie er diese umfahren kann oder schnellstmöglich aus ihnen heraus kommt. Der bei diesem Studienprojekt erstellte Routing Service ist sehr einfach ausgestattet. Es kann nur eine Start- und Zielposition (Koordinaten) in einer Anfrage angegeben werden. Bei der Implementierung des Routing Service und der anderen Dienste wurden keine Spezifikationen beachtet. Des Weiteren erhält der Nutzer vom Routing Service nur geometrische Informationen zur Route, keine Fahranweisungen oder Karten.

1.3 Anmerkungen zum Aufbau der Diplomarbeit

Die Diplomarbeit gliedert sich in einen Hauptteil mit 7 Kapiteln und einen Anhang aus 3 Teilen.

Kapitel 1.: Einleitung - Dieses Kapitel

Kapitel 2.: OpenLS - Vorstellung der OpenLS Spezifikation

Kapitel 3.: Grundlagen - Vorstellung der verwendeten Techniken und Programme

Kapitel 4.: Prototypische Implementierung des Route Service - Beschreibung der konkreten Umsetzung des Route Services

Kapitel 5.: Zusätzliche Implementierungen - Vorstellung von möglichen Implementierungen eines Emergency Route Service und einer graphischen Benutzerschnittstelle für die Nutzung des Route und Emergency Route Services

Kapitel 6.: Installationshinweise - Beschreibung der Installation verschiedener Services und Datenbanken

Kapitel 7.: Zusammenfassung - Abschließende Betrachtung der Arbeit und Ausblick

A Anhang: - Weitere Informationen

- **A.1** - Beispiel für GetMap Request an Geoserver WMS
- **A.2** - UML-Klassendiagramme
- **A.3** - Inhaltsverzeichnis CD

2 OpenLS™ Spezifikation

Im Folgenden soll das *Open Geospatial Consortium* (OGC) vorgestellt werden, welches als Organisation hinter der OpenLS Spezifikation steht. Danach wird die Spezifikation mit ihrer Architektur und den Aufgaben der einzelnen Komponenten vorgestellt.

2.1 OGC

OGC ist die Abkürzung für *Open Geospatial Consortium* (früher: Open GIS[1] Consortium). Dieses hat sich zum Ziel gesetzt, die Entwicklung von raumbezogener Informationsverarbeitung (insbesondere Geodaten) auf Basis allgemeingültiger Standards zum Zweck der Interoperabilität festzulegen. Sie wurde 1994 von 20 Mitgliedern gegründet und ist eine gemeinnützige Organisation. Heute sind es über 300 Mitglieder, die sich aus Regierungsorganisationen, privater Industrie und Universitäten zusammensetzen. Die Mitgliedschaft im OGC ist kostenpflichtig und das registrierte Markenzeichen ist OpenGIS.

Die Entwicklung offener Standards beruht auf der Basis frei verfügbarer Spezifikationen, die von abstrakten Beschreibungen des Aufbaus, der Komponenten und der Funktionsweise eines dienstebasierten GIS im Sinne des OGC bis hin zu detaillierten Spezifikationen der Implementation dieser Dienste reichen. Hierbei wird jedoch nicht die konkrete Umsetzung der Software vorgeschrieben, sondern die verschiedenen Schnittstellen eines Dienstes, deren Eigenschaften und Verhalten festgelegt.

Der Weg zu diesen Spezifikationen läuft über einen langen Diskussionsprozess im OGC, dessen Ergebnis schließlich in einer „Spezifikation“ resultiert.[18][34]

[1] Geographisches Informationssystem - GIS

2.2 OpenLS

OpenLS ist die Abkürzung für *Open Location Services* (oder auch *OpenGIS Location Services*)[25]. Sie ist eine Initiative des *Open Geospatial Consortiums* (OGC), welche im Jahr 2000 ins Leben gerufen wurde und sich der Entwicklung und Beschreibung von Schnittstellen und Protokollen widmet, um *Location Based Services* zu standardisieren. Das Ziel dieser Initiative ist es, freie Spezifikationen für interoperable *Location Based Services* zu erarbeiten und Entwicklern zur Verfügung zu stellen. Diese freien Spezifikationen sollen den Gebrauch von der Position des Nutzers mit weiteren räumlichen Informationen und erforderlichen Methoden in der drahtlosen Internet-Umgebung erleichtern. Bei der Erstellung der OpenLS Spezifikation wurden bereits bestehende Spezifikationen und Infrastrukturen von Telekommunikations- und Internetservices beachtet und mit einbezogen.

Die OpenLS Spezifikation kann als Plattform angesehen werden, die als Basis für verschiedene Dienste aus dem Bereich der Location Based Services genutzt werden kann.

> *Location Based Services* (LBS)
> Standortbezogene (ortsbasierende-) Dienste sind über ein Netzwerk erbrachte Dienste, die unter Zuhilfenahme von positions-, zeit- und personenabhängigen Daten dem Endbenutzer selektive Informationen bereitstellen oder Dienste anderer Art erbringen.[2]

Einfache Beispiele für LBS wären Navigations- und Informationsdienste. Eine Anfrage an solche Dienste könnte wie folgt aussehen:

- „Wo finde ich um meine aktuelle Position herum das nächste Restaurant?“
  Informationsdienst
- „Und wie komme ich von meiner Position dorthin?“
  Navigationsdienst

Die OpenLS Spezifikation enthält mehrere Beschreibungen für LBS. Sie können als eine detaillierte technische Beschreibung der Schnittstellen bezeichnet werden oder auch als Bauplan für OpenLS konforme Web Services. Die Beschreibung dieser Web Services erfolgt durch eindeutig spezifizierte *Abstract Data Types* (ADT). Für Anfragen und Antworten an die Web Services werden diese in das XML-Format *XLS* (*XML for Location Services*, nicht zu verwechseln mit *Mircosoft Excel - XLS*) kodiert. Die ADTs und XLS werden später noch ausführlicher erläutert.

[2] aus Wikipedia[34]

Die verschiedenen Web Services (LBS) werden als so genannte *Core Services* (dt. Kerndienste) bezeichnet und unterteilen sich in folgende fünf Punkte:

- *Part 1:* **Directory Service**
- *Part 2:* **Gateway Service**
- *Part 3:* **Location Utility Service**
- *Part 4:* **Presentation Service**
- *Part 5:* **Route Service**

Diese fünf Core Services bilden das Service-Framework. Das Service-Framework und die erwähnten ADTs bilden zusammen den *GeoMobility Server* (GMS), die OpenLS-Plattform.

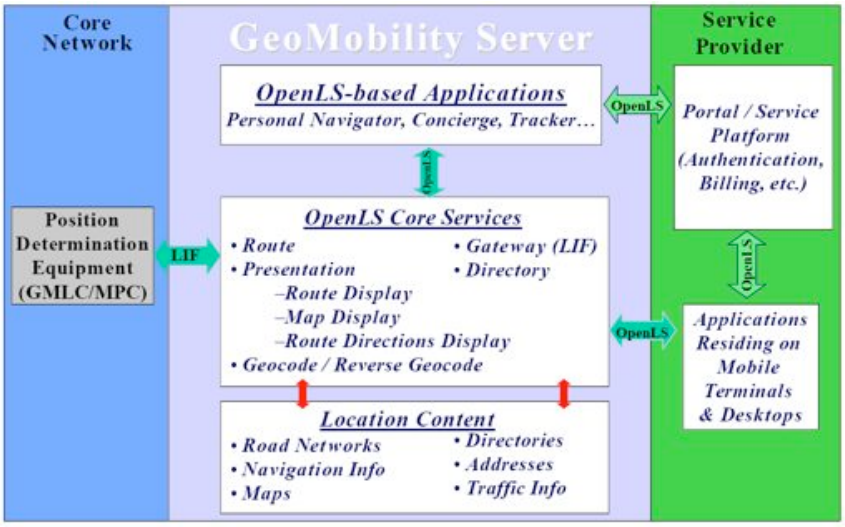


Abbildung 2.1: The GeoMobility Server (OpenLS 2005)

Zusammenfassend enthält der GeoMobility Server:

- die Core Services und ihre OpenLS-Schnittstellen;
- das OpenLS Informationsmodell, das sich aus den ADTs zusammensetzt;
- evtl. eine Reihe lokaler Anwendungen, die auf Core Services aufbauen und über die OpenLS-Schnittstellen zugreifen;
- Inhalte wie Kartenmaterial, POIs[3], Routen etc., welche durch die Core Services verwendet werden. Die Inhalte können auch auf anderen Servern liegen und über das Internet abgerufen werden;

[3]Points of Interest (POI) - Interessante Punkte, z.B. Sehenswürdigkeiten

- evtl. andere unterstützende Funktionen für Personalisierung, Context-Management, Gebührenerhebung, Protokollierung etc.

Nachfolgend werden die fünf Core Services beschrieben.

2.2.1 Part 1: Directory Service

Der *Directory Service* ermöglicht einen netzwerkbasierten Zugriff auf Online-Verzeichnisse, wie z.B. „Gelbe Seiten“ (Online-Branchenverzeichnis). Mit den Verzeichnissen können Orte, Produkte oder Dienste gesucht werden, die sich entweder um die eigene Position herum befinden oder deren Position gesucht wird.

Mit Hilfe einer OpenLS konformen Anwendung kann der Nutzer eine Anfrage (Request) an den Dienst (Service) stellen. Die Anwendung erstellt dabei aus der Suchanfrage des Nutzers einen XLS-Request und sendet ihn an den Service. Der Service empfängt diesen Request, liest ihn aus und bearbeitet ihn.

Bei einem Request an einen Directory Service, muss zwischen zwei Arten unterscheiden werden:

- *Pinpoint Directory Service*
  Suche nach einem Ort, Produkt oder Dienst, der nichts mit der aktuellen Position zu tun hat, z.B. „Suche Nassauer Hof in Wiesbaden!“
- *Proximity Directory Service*
  Suche des nächsten Ortes, Produktes oder Dienstes mit Hilfe der aktuellen Position, z.B. „Suche nächstes Hotel um meine aktuelle Position herum!“

Häufig ist es aber nicht möglich den Request zwischen diesen beiden Arten zu unterscheiden, da meistens eine Mischung aus diesen beiden Arten vorkommt (z.B. bestimmter Dienst in Umgebung: „Suche nächstes McDonalds Restaurant im Umkreis von 2 Kilometern!“).

Die Antwort (Response) des Directory Service kann auch mehrere Treffer auf eine Anfrage enthalten. Es ist jedoch möglich, bereits beim Request an den Service weitere einschränkende Suchkriterien mit anzugeben. Diese Parameter werden dann vom Service beim Erstellen der Response beachtet (z.B. „nächstliegendes Hotel an erster Stelle, maximal 3 Ergebnisse“ oder „alphabetisch absteigend sortiert“).

2.2.2 Part 2: Gateway Service

Beim Gateway Service handelt es sich um einen Dienst, mit dessen Hilfe es möglich ist, die aktuelle Position eines Nutzers zu erfahren. Dies geschieht durch einen netzwerkbasierten Zugriff auf die Positionsdaten eines bekannten mobilen Gerätes (z.B. Mobiltelefon). Dieser Service ist somit eine Schnittstelle zwischen dem GeoMobility Server und einem Location Server. Ein Location Server wird von Mobilfunknetzbetreibern zur Verfügung gestellt und kann die Position eines mobilen Gerätes innerhalb des zugehörigen Mobilfunknetzes berechnen. Das benutzte Interface entspricht der *Mobile Location Protocol Specification* (MLP)[17][16]. Sie ist eine *Application Programming Interface* (API - dt. Programmierschnittstelle) und definiert den Zugriff auf die Positionsdaten mobiler Geräte, unabhängig vom darunter liegenden Netzwerk, so dass Positionsdaten des Nutzers eines beliebigen Telefonnetzwerkes (z.B. GSM und UMTS) zwischen dem Gateway Service und dem Location Server ausgetauscht werden können.

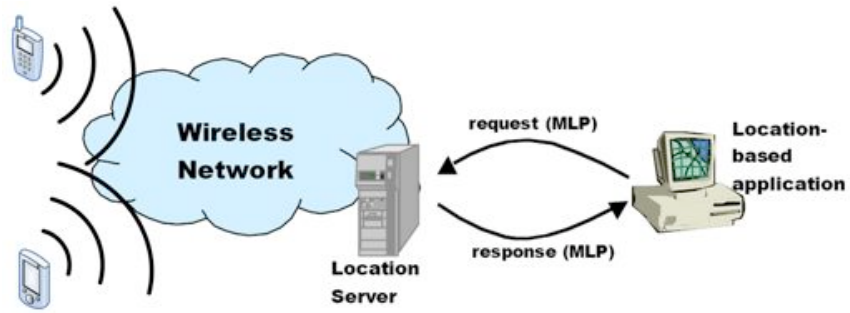


Abbildung 2.2: Zugriff auf Positionsdaten mit MLP (geändert, aus MLP[17])

2.2.3 Part 3: Location Utility Service

Der Location Utility Service unterteilt sich in zwei Services: den *Geocoder Service* und den *Reverse Geocoder Service*. Die Aufgaben der beiden Services sind vergleichbar.

Der Geocoder Service gibt auf Grund eines Namens oder einer Adresse (normalisierte Beschreibung) eine geographische Position (Koordinaten) zurück. Die Beschreibung dieser Position erfolgt in der *Geography Markup Language* (GML)[19] vom OGC. Der Service antwortet mit einer Liste von Adressen und den dazugehörigen Positionen (Koordinaten), sortiert nach der besten Übereinstimmung mit der übergebenen Adresse. Des Weiteren ist er somit in der Lage, eine nicht vollständige Adresse zu vervollständigen und diese zurückzugeben.

Der Reverse Geocoder macht das Ganze in umgekehrter Reihenfolge. Ihm wird eine geographische Position (Koordinate) übergeben und er antwortet mit einer normalisierten Beschreibung (Straßenadresse). Dem Request an den Reverse Geocoder kann ein optionaler Parameter eingefügt werden, mit welchem es möglich ist, die Adressform des Service Res-ponses auszuwählen. Als Antwort zu der Position wären z.B. nahe liegende Kreuzungen oder POIs möglich. Wird kein zusätzlicher Parameter angegeben, antwortet der Service standardmäßig mit der am nähsten liegenden Adresse.

2.2.4 Part 4: Presentation Service

Dieser über das Netzwerk erreichbare Dienst ist für die grafische Präsentation von Karten zuständig. Jedes OpenLS konforme Anwendungsprogramm kann diesen Service nutzen. Er ist vergleichbar mit eine OGC *Web Map Service*[4] (WMS), unterscheidet sich jedoch in ein paar Punkten von ihm. Parameter, wie Mittelpunkt und Radius oder Displaymaßstab, können optional angegeben werden. Auch ist es möglich, bei einer Karten-Anfrage eine Bildschirmauflösung zu berücksichtigen. Die Karte wird über die angegebene „BoundingBox“ und allen verfügbaren oder den angegebenen Layern erstellt. Optional können in der Karte die mitgelieferten ADTs dargestellt werden, möglich wären z.B. POIs oder AOIs[5]. Es könnten aber auch Adressen oder eine berechnete Route incl. deren Richtung, Manöver und Beschreibung dargestellt werden.

2.2.5 Part 5: Route Service

Mit einer der wichtigsten Komponente im OpenLS-Framework ist der Route Service. Da die Implementierung dieses Route Services einen großen Teil dieser Diplomarbeit ausgemacht hat, wird später noch ausführlicher auf ihn und seine Möglichkeiten eingegangen.

Der Route Service ist ebenfalls ein netzwerkbasierter Service, der für einen Route Request von einer OpenLS konformen Applikation eine Fahr- oder Gehroute ermittelt. Dabei gibt es zwei verschiedene Arten von Route Requests. Zum einen eine neue Anfrage zu einer Route, welche die Berechnung einer Route zur Folge hat. Zum anderen gibt es die Möglichkeit, eine bereits berechnete Route, die auf dem Route Service gespeichert wurde, wieder abzurufen oder neu berechnen zu lassen. Es können natürlich nur solche Routen abgerufen werden, die vorher auf dem Route Service gespeichert wurden. Das Speichern

[4] WMS - stark vereinfacht: Ein Programm, welches im Internet auf einem Server läuft und auf Anfrage gewünschte Karten erstellen und zurücksenden kann. Eine genauere Beschreibung erfolgt im Kapitel 3.4.3.1.

[5] Area of Interest

einer Route muss aber explizit bei einem Route Request, der eine Neu-Berechnung einer Route zur Folge hat, mit angegeben werden.

Bei einem neuen Route Request müssen mindestens Start- und Zielpunkt sowie die Route-Eigenschaft, z.B. „Schnellste“, „Kürzeste“ oder „Zu Fuß“, angegeben werden. Alle anderen Parameter, wie z.B. Zwischenpunkte, Vermeidungsgebiete, Fahranweisungen, Karten, Geometrien etc., sind optional und werden später noch ausführlicher behandelt. Ein Response von einem Route Service besteht mindestens aus einer Route-Zusammenfassung. Diese enthält Informationen, z.B. über Reisezeit und Reisestrecke. Optional kann der Client die eben erwähnten Fahranweisungen, Karte(n) und Geometrien der Route erhalten.

2.3 OpenLS Information Model

Wie schon zuvor beschrieben (Kapitel 2.2), werden die einzelnen Web Services als so genannte *Core Services* bezeichnet. Die *Abstract Data Types* (ADT) spielen im Zusammenhang damit eine wichtige Rolle. Sie sind wie folgt in der OpenLS Spezifikation definiert:

> *Abstract Data Type* (ADT)
> The basic information construct used by the GeoMobility Server and associated Core Services. Consists of well-known data types and structures for location information. Defined as application schemas that are encoded in XML for Location Services (XLS).[25]

Für die Kommunikation zwischen diesen Services (GMS) und mit den Clients werden die ADTs genutzt. Damit die ADTs über das Netzwerk mit den verschiedenen Services und Clients verwendet werden können, werden sie in *XML for Location Services* (XLS) codiert. XLS definiert eine Methode für die Codierung und Decodierung einer Anfrage/Antwort an einen Core Service und der dazugehörigen ADTs des GMS.

XLS basiert auf XML und soll eine verständliche Darstellung von Geoobjekten mit minimalem Overhead und minimaler Komplexität ermöglichen (d.h., einfach zu codieren und zu decodieren). Da mobile Geräte durch geringe Kapazitäten (Bandbreite, Display, Speicherplatz) gekennzeichnet sind, soll XLS auf die unterschiedlichen Kapazitäten skalierbar gehalten werden, ähnlich den Versionen „Tiny“ und „Basic“ der *Scalable Vector Graphics* (SVG). Um diese und weitere Forderungen zu erfüllen, wurden XML Schemata entworfen, welche die ADTs darstellen und erweitern können.[3]

Eine Übersicht über die ADTs und ihre Nutzung stellt die folgende Abbildung dar, das OpenLS Information Model.

The OpenLS Information Model

Position ADT	Point location in well-known coordinate system
Address ADT	Street address or intersection
Point of Interest (POI) ADT	The location where someone can find place, product or service
Area of Interest (AOI) ADT	A polygon, bounding box or circle used as a search template
Location ADT	A location (Position, Address or POI)
Map ADT	The portrayal of maps and feature overlays (routes & POI)
Route Summary ADT	Metadata pertaining to a route
Route Geometry ADT	Geometry data for a route
Route Maneuvers ADT	Navigation maneuver data for a route
Route Directions ADT	Turn-by-turn navigation instructions for a route

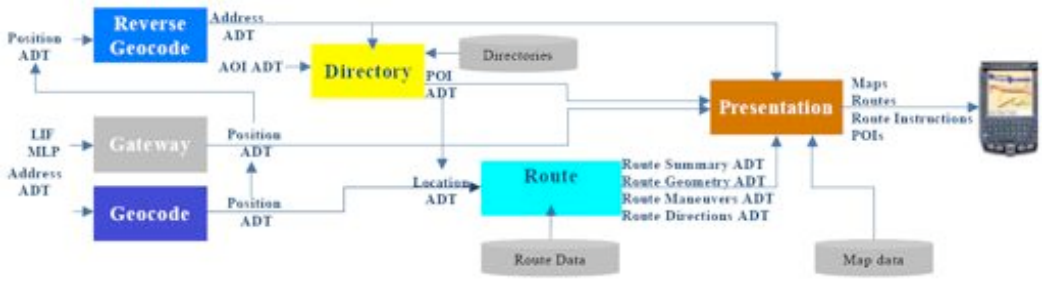


Abbildung 2.3: The OpenLS Information Model (OpenLS 2005)

Die Beschreibung der einzelnen ADTs fällt in der vorherigen Abbildung 2.3 sehr kurz aus, dabei sind sie vielseitiger als es die Übersicht in der Abbildung zeigt. Aus diesem Grund soll hier kurz der Inhalt des *Route Instructions List* ADTs beschrieben werden.

Die *Route Instructions List* enthält eine Liste von Fahr- oder Gehanweisungen, bestehend aus „Schritt für Schritt“-Richtungsanweisungen entlang der Route. Sie kann mit Hilfe des Presentation Service dem Nutzer optisch dargestellt werden.

Die nächste Abbildung zeigt den Ablauf einer Anfrage an und die dazugehörige Antwort von einem Web Service.

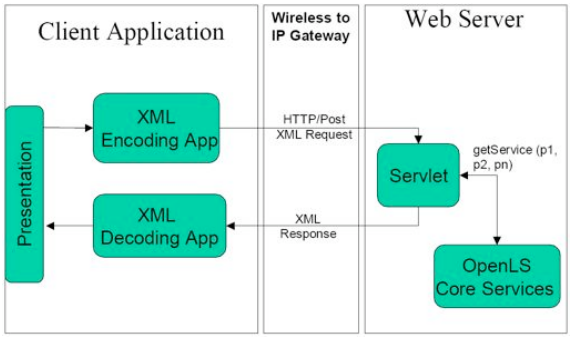


Abbildung 2.4: Ablaufmuster für OpenLS Request/Response Pair (OpenLS 2005)

Im Web Service Kontext wird dann von einem Request und Response Pair (dt. Anfrage und Antwort Paar) gesprochen. Die Client-Anwendung codiert den Request für den Core Service als einen XML Request. Mit Hilfe der HTTP-POST Methode wird der XML Request an ein Servlet geschickt. Dieses Servlet analysiert den XML Request, welcher entsprechend des angefragten Core Services an diesen weitergeleitet wird. Der Core Service verarbeitet den Request und schickt den Response zum Servlet zurück. Das Servlet codiert die Antwort als XML Response und leitet sie an die Client-Anwendung weiter. Diese decodiert den XML Response und zeigt die angefragten Daten auf dem Display des Nutzers an.

Auf die Funktionsweise des *Hypertext Transfer Protocol* (HTTP) und die verschiedenen Methoden, die damit verbunden sind, wird in Kapitel 3.4.3.3 noch einmal ausführlicher eingegangen.

3 Grundlagen

In diesem Kapitel werden für das weitere Verständnis grundlegende Begriffe und Techniken eines Routenplaners und des zu implementierenden Route Services vorgestellt und beschrieben. Das Ziel dabei ist es, eine Übersicht zu vermitteln. Bei Interesse kann in den angegeben Referenzen die Thematik weiter vertieft werden.

3.1 Routenplaner

3.1.1 Tour/Route/Routing

Der Begriff *Tour* wird für Ortsveränderungen verwendet, deren Ausmaße groß und deren Ausgangs- und Zielpunkt in der Regel identisch sind. Aus diesem Grund muss zwischen Tourenplaner und Routenplaner unterschieden werden.

Ein Tourenplaner soll z.B. einem Speditions-Unternehmen helfen, ihre Touren zu planen. Dabei soll bei der Planung, z.B. für einen LKW, eine optimale Reihenfolge der Ziele gefunden werden. Die Software könnte folgende Ziele haben: Minimierung der eingesetzten Fahrzeuge, des Personals, der zurückgelegten Strecke oder der benötigten Zeit.

Ein Routenplaner hingegen ist ein Programm, mit dessen Hilfe eine Route durch Angabe eines Start- und Zielpunktes unter Berücksichtigung bestimmter Kriterien berechnet werden kann. Ein Kriterium könnte z.B. sein, die schnellste Route zu ermitteln. Des Weiteren kann es auch möglich sein sogenannte Zwischenpunkte anzugeben. Diese werden dann in der Regel in der angegebenen Reihenfolge angefahren.

Obwohl meine Diplomarbeit nichts mit der Elektrotechnik zu tun hat, bediene ich mich einer Definition aus dem Bereich der Nachrichtentechnik[1] zum Begriff *Routing*. Nach dieser bedeutet *Routing* die Auswahl der Wegeabschnitte beim Aufbau von Nachrichtenverbindungen in einem Netzwerk, die unter Berücksichtigung von Kriterien, wie kürzeste Entfernung etc., erfolgen kann. Bezogen auf meine Diplomarbeit bedeutet dies: „Ermitt-

[1] Nachrichtentechnik ist ein Teilgebiet der Elektrotechnik, welche sich mit den technischen Aspekten der Übertragung und Verarbeitung von Informationen beschäftigt

lung der Strecke (Streckenabschnitte) zwischen Start- und Zielpunkt in einem Graphen, unter Berücksichtigung von vorgegebenen Kriterien.“ Es sollen allerdings keine Nachrichten versendet, sondern die bestmöglichste Strecke (unter Berücksichtigung bestimmter Kriterien) für ein Fahrzeug oder einen Fußgänger ermittelt werden. Beim Routing in der Nachrichtentechnik geschieht dies auf Basis von Netzwerken, bei einem Routenplaner auf Basis eines Graphen, welcher nun folgend erklärt wird.

3.1.2 Graph

Um den eben erklärten Begriff *Routing* in einem Programm umsetzen zu können, werden zwei Dinge benötigt. Zum einen ein *Routing-Algorithmus*, der z.B. die Ermittlung des kürzesten Weges übernimmt und zum anderen ein *Graph*, auf dem geroutet werden kann.

Ein *Graph* ist laut der Graphentheorie ein Gebilde aus Knoten (Ecken oder Punkte), die durch Kanten (Linien) verbunden sein können. Unter der Graphentheorie versteht sich ein Teilgebiet der Mathematik, welches die Eigenschaften und Beziehungen von Graphen zueinander untersucht. Der einzige Unterschied zwischen gewichteten und ungewichteten Graphen liegt bei der Angabe der Wichtung einer Kante. Die Wichtung einer Kante könnte z.B. die Distanz oder Fahrdauer zwischen zwei Knoten sein.

Um den Begriff *Graph* etwas vereinfacht zu erklären, kann dieser mit einem Liniennetzplan verglichen werden. Liniennetzpläne befinden sich z.B. in Straßenbahnen oder an Bushaltestellen. Sie sagen etwas darüber aus, welche Buslinien welche Haltestellen anfahren. Folgende Abbildung zeigt einen Ausschnitt des Liniennetzplanes der Mainzer Verkehrsgesellschaft (MVG)[2].

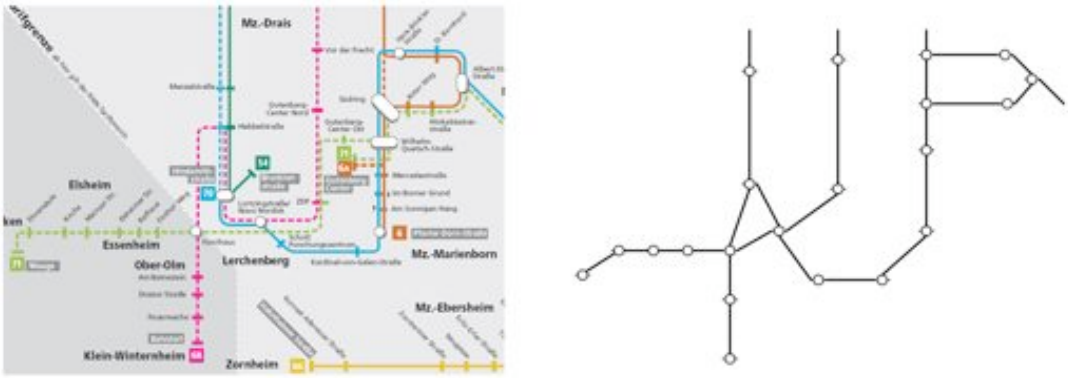


Abbildung 3.1: Liniennetzplan 2006 MVG / Resultierender Graph (grob skizziert)

Wenn die Haltestellen als Knoten (Kreuzungen, Punkte), die Verbindungslininen zwischen den Haltestellen als Kanten (Straßenabschnitte, Linien) und die Fahrdauer zwi-

[2]Mehr Informationen unter: http://www.mvg-mainz.de

schen den Haltestellen als Wichtung (Gewicht) bezeichnet werden, handelt es sich vereinfacht gesagt um einen gewichteten Graphen. Er repräsentiert somit ein Straßennetz, welches verschiedene Punkte über Linien mit unterschiedlichen Wichtungen miteinander verbindet. Da für das Routing ein gewichteter Graph erforderlich ist, wird ab hier bei der Erwähnung von Graphen von gewichteten Graphen ausgegangen.

Bei gewichteten Graphen wird zwischen fünf verschiedenen Typen unterschieden:

- ungerichtete Graphen ohne Mehrfachkanten (Abbildung 3.2 Graph 1)
- gerichtete Graphen ohne Mehrfachkanten (Abbildung 3.2 Graph 2)
- ungerichtete Graphen mit Mehrfachkanten (Abbildung 3.2 Graph 3)
- gerichtete Graphen mit Mehrfachkanten (Abbildung 3.2 Graph 4)
- Hypergraphen
  Als Hypergraph werden Graphen bezeichnet, bei dem Kanten mehr als nur zwei Knoten verbinden können. Da die Darstellung solcher Graphen-Typen nur schwer abbildbar ist, soll hier nicht weiter darauf eingegangen werden.

Die Erklärung der Typen lässt sich am Besten anhand der Abbildungen aufzeigen.

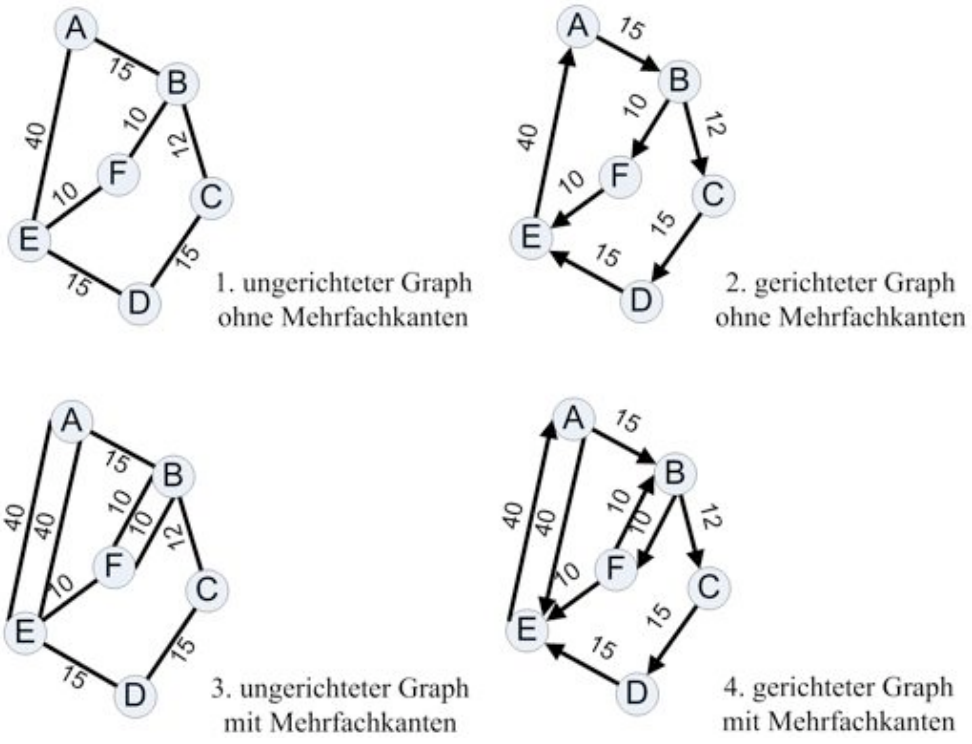


Abbildung 3.2: Graph-Typen

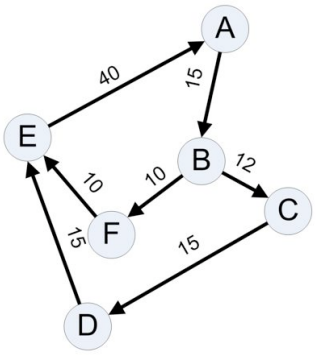
Die allgemeinen Definitionen lauten:

- Der Graph wird bezeichnet mit $G = (V, E)$,
- die Knoten sind dabei $V(G) = \{v_i | i = 1, ..., n\}$ und
- die Kanten werden durch $E(G) = \left\{e_{ij} =< v_i, v_j > | v_i, v_j \in V(G)\right\}$ dargestellt.
- Eine Kante $e_{ij} \in E(G)$ besitzt die Wichtung $w_{ij} := w(e_{ij} \in \mathbb{R}^+)$

Für diese Diplomarbeit werden nur gerichtete Graphen ohne Mehrfachkanten (in Abbildung 3.2 auf der vorherigen Seite Graph 2) genutzt. Die Definitionen werden nun beispielhaft anhand dieses Graphen aufgezeigt:

Der Graph besteht aus den Knoten $V(G) = \{v_A, v_B, v_C, v_D, v_E, v_F\}$,
den Kanten $E(G) = \{e_{AB}, e_{BC}, e_{CD}, e_{DE}, e_{FE}, e_{EA}, e_{BF}\}$ und
den Wichtungen $w(E) = \{w_{AB} = 15, w_{BC} = 12, w_{CD} = 15, w_{DE} = 15, w_{FE} = 10\}$ und
$\{w_{EA} = 40, w_{BF} = 10\}$.

Ebenfalls ist es wichtig zu erwähnen, dass der Knoten eines Graphen im Normalfall keine geometrischen Informationen (wie z.B. die Koordinaten) enthält. Um einen Graphen übersichtlicher darzustellen, können die Kanten als Gummifäden und die Knoten als frei verschiebbar angesehen werden. Zum besseren Verständnis zeigt die folgende Abbildung den für diese Diplomarbeit wichtigen Graphen-Typ 2 von der Abbildung 3.2 auf der vorherigen Seite in „geänderter Form“.



Wie sehr gut zu sehen ist, hat der Graph seine ursprüngliche Form verändert. Knoten befinden sich an anderen Orten und Kanten sind länger oder kürzer geworden. Die grundlegende Topologie des Graphen wurde aber nicht verändert. Mit Topologie ist ein Teilgebiet der Mathematik gemeint. Es beschäftigt sich mit der Untersuchung der Eigenschaften geometrischer Gebilde, die durch Umformungen, wie Dehnen, Stauchen, Verbiegen, Verzerren, Verdrillen etc., eines Gegenstandes nicht verändert werden.

Für den Graphen bedeutet, egal wie er verformt wird durch dehnen, verbiegen etc., die Topologie ist immer dieselbe. Knoten A ist immer mit Knoten B und E verbunden und die Kante AB oder Kante CD werden immer ihre Wichtungen von 15 enthalten.

3.1.3 Routing-Algorithmus

Wie eben erwähnt, benötigt ein *Routing-Algorithmus* einen Graphen, um einen Weg zwischen dem Start- und Zielpunkt ermitteln zu können.

Der Begriff *Routing-Algorithmus* stammt ebenfalls wie der Begriff *Routing* aus dem Bereich der Nachrichtentechnik. Er kann wie folgt beschrieben werden: Mit Hilfe von Algorithmen und Entscheidungskriterien (z.B. kürzeste Route, schnellste Route) werden Werte errechnet, die eine Aussage über ein Maß für die Optimalität einer Route geben.

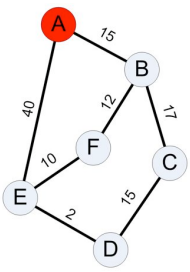
Es gibt eine ganze Reihe von Algorithmen, die in Routenplanern verwendet werden können, wie z.B.: A*-, Floyd-Warshall- oder Bellman-Ford-Algorithmus. Der wohl bekannteste Algorithmus ist jedoch der Dijkstra-Algorithmus, welcher auch in GeoTools[3] implementiert ist und daher für meine Diplomarbeit von großer Bedeutung ist.

Dijkstra-Algorithmus

Der Algorithmus von Dijkstra (nach seinem Erfinder Edsger W. Dijkstra[5] benannt) bestimmt ausgehend vom Startknoten den kürzesten Pfad (engl. „Shortest Path“) zu allen Nachbarn und deren Nachbarn bis zu einem beliebigen Knoten (Zielknoten) in einem kantengewichteten Graphen. Die Wichtungen der Kanten dürfen dabei nicht negativ sein, ansonsten kann es zu Problemen bei der Ermittlung der Route kommen.

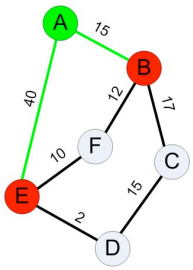
Die Wichtung einer Kante sagt dabei etwas für deren Bewertung aus. Sie kann z.B. unterschiedlicher Herkunft sein. Soll z.B. der kürzeste Weg ermittelt werden, muss die Wichtung der Kante den Wert der Distanz zwischen den beiden Knoten annehmen. Soll der schnellstmögliche Weg ermittelt werden, muss die Wichtung den Wert der Fahrdauer zwischen den beiden Knoten erhalten. Das im Falle der schnellstmöglichen Route aber weitere Probleme auftreten können, wird später bei der konkreten Implementierung noch erwähnt werden. Denn: *Nicht immer ist der scheinbar schnellste, auch der beste Weg!*

Die grobe Funktionsweise des Dijkstra-Algorithmus soll nun hier an einem kleinen Graphen veranschaulicht werden. Gesucht ist der kürzeste Weg von A nach D:



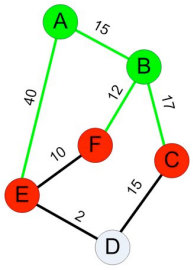
Ausgangsgraph:
Als erstes wird der Graph initialisiert: Alle Knoten sind grau, alle Kanten schwarz. Der Startknoten wird rot gefärbt.

[3] GeoTools - The open source Java GIS toolkit ; Mehr in Kapitel 3.4.2.3



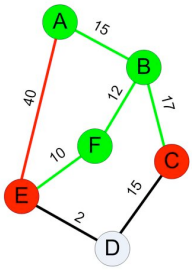
1. Durchlauf:

Der Startknoten wird grün gefärbt. Seine grauen Nachbarn B und E werden rot gefärbt. Sie befinden sich jetzt am Rand des untersuchten Teilgraphen. Die zu B und E führenden Kanten werden grün, da es die kürzesten Wege zu ihnen sind.



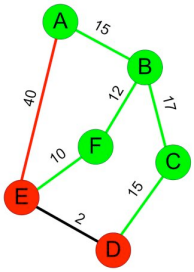
2. Durchlauf:

Der Knoten B ist nun der rote Knoten mit minimalem Abstand zu A. Er wird grün gefärbt. Gleichzeitig werden seine Nachbarn C und F rot. Die Kante zu C wird grün, da sie auf dem bislang kürzesten Weg zu C liegt. Die Kante zu F wird ebenfalls grün, da sie ebenfalls auf dem kürzesten Weg zu F liegt.



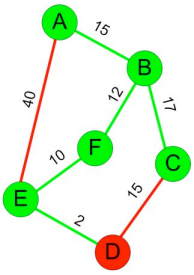
3. Durchlauf:

Der Knoten F wird mit der minimalsten Entfernung von 27 zu Knoten A grün gefärbt. Knoten E wird rot und die Kante zu E wird grün gefärbt. Daraus ergibt sich eine neue kürzeste Entfernung zu E von 37. Damit wird die Kante von A nach E rot eingefärbt, deren Entfernung beträgt 40.



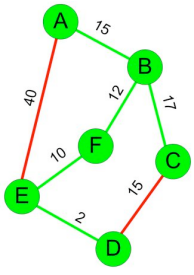
4. Durchlauf:

Minimal ist Knoten C von Knoten A mit einer Entfernung von 32. Damit wird Knoten C, die Kante zwischen C und D grün und Knoten D rot.



5. Durchlauf:

Minimal ist Knoten E von Knoten A mit einer Entfernung von 37. Damit wird Knoten E, die Kante von E nach D grün eingefärbt. Weil die kürzeste Entfernung von A nach D über B,F und E geht, wird die Kante zwischen C und D rot eingefärbt.



6. Durchlauf:

Schließlich verbleibt nur noch Knoten D, welcher abschließend grün gefärbt wird, da keine roten Nachbarknoten mehr zu betrachten sind. Die kürzesten Entfernungen von Knoten A zu allen anderen Knoten lassen sich nun aus dieser Abbildung entnehmen.

Der kürzeste Weg von Knoten A zu Knoten D verläuft somit über die

Knoten B,F und E mit einer Entfernung von 39. Allerdings bleibt die Frage, ob dies auch der bestmöglichste Weg ist. Wohl eher nein, denn der Weg von Knoten A zu Knoten D könnte genauso gut nur über E verlaufen. Hier wäre zwar dann die Entfernung 42, also um drei Einheiten höher, aber der Weg hätte nur einen Knoten. Das Problem *kürzester vs. schnellster Weg* wird später noch bei der Implementierung im Kapitel 4.3.3.1 angesprochen.

3.1.4 Verschneidung

Damit gesperrte Straßen oder gefährdete Gebiete bei einer Routenplanung berücksichtigt, sprich umfahren werden können, müssen diese von der Planung der Route ausgeschlossen werden. Ist nur eine Straße zu sperren, ist die Lösung nicht all zu schwer. Problematischer wird es, wenn ein Gebiet nicht durchfahren werden soll. Hier muss eine Verschneidung von Geometrien erfolgen, um die nicht befahrbaren Straßen zu ermitteln. Unter Geometrien sind bei den Gebieten Polygone und bei den Straßen Linien zu verstehen. Genauer gesagt muss eine Verschneidung zwischen einem (oder mehreren) Polygon(en) und dem Straßennetz erfolgen. Beim Verschneiden von Geometrien gibt es innerhalb meiner Diplomarbeit vier mögliche Lösungen, welche nachfolgend aufgezählt und in der Abbildung gezeigt werden:

- die Straße liegt *innerhalb* (1)
- die Straße liegt *außerhalb* (2)
- die Straße *schneidet* den gesperrten Bereich (3)
- die Straße *berührt*, schneidet aber NICHT den gesperrten Bereich (4)

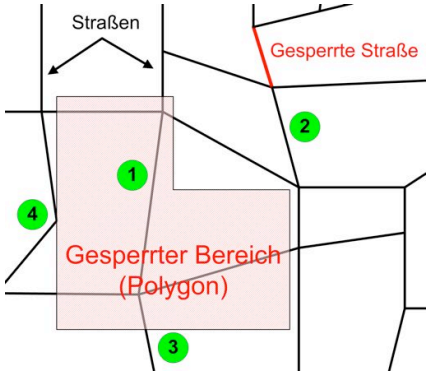


Abbildung 3.3: Verschneidung Straßennetz/gesperrter Bereich

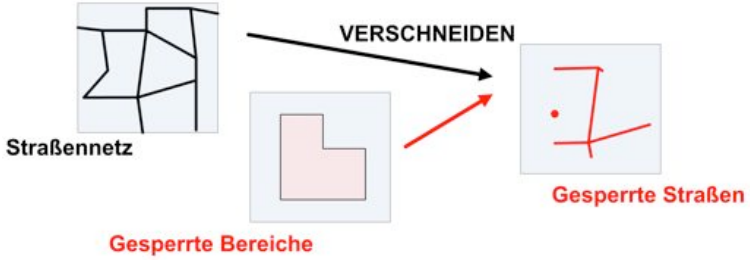


Abbildung 3.4: Ablauf einer Verschneidung - Straßennetz mit gesperrten Bereichen

3.2 JAVA

Java ist eine von der Firma Sun Microsystems[30] entwickelte Software-Technologie. Es wird versucht, diese hier kurz vorzustellen. Bei größerem Interesse kann in den angegebenen Referenzen weiter gelesen werden. Um die Java-Technologie genauer erklären zu können, unterteile ich sie zu Beginn in folgende Komponenten:

3.2.1 Programmiersprache Java

3.2.2 Java Virtual Maschine (Java VM)
Übersetzt Java-Programme und führt diese aus.

3.2.3 Java Application Programming Interface (Java API)
Bibliothek mit einer großen Ansammlung von Klassen.

Auf diesen Komponenten aufbauend gibt es folgende vier Technologien:

- *Java 2 Platform Standard Edition* (Java SE, vormals J2SE)
  Mit Hilfe der Java SE können Java Anwendungen für Desktop-Rechner und Server entwickelt und ausgeführt werden. Java SE beinhaltet des Weiteren Klassen für die Entwicklung von Web Services und stellt die Grundlage für Java EE zur Verfügung.[4]

- *Java 2 Platform Enterprise Edition* (Java EE, vormals J2EE)
  Die Java EE ist ein Industriestandard, um Server-seitige Java Anwendungen zu entwickeln.[5]

- *Java 2 Platform Micro Edition* (Java ME, vormals J2ME)
  Dies ist die Grundlage, um Java-Anwendungen für Mobiltelefone oder PDAs umzusetzen.[6]

- *Java Card*
  Java Card-Technologie stellt eine Umgebung zur Verfügung, die es ermöglicht, Chipkarten-Anwendungen zu erstellen.[7]

[4] Mehr Informationen unter: http://java.sun.com/javase
[5] Mehr Informationen unter: http://java.sun.com/javaee
[6] Mehr Informationen unter : http://java.sun.com/javame
[7] Mehr Informationen unter: http://java.sun.com/products/javacard

3.2.1 Programmiersprache Java

Im Frühjahr 1991 bis Sommer 1992 wurde die Urversion von Java (damals Oak) im Auftrag des US-amerikanischen Computerherstellers Sun Microsystems entwickelt. Das eigentliche Ziel war es nicht nur eine neue Programmiersprache zu entwickeln, sondern viel mehr noch die Entwicklung eines neuen vollständigen Betriebssystems. Dieses Betriebssystem sollte kleinen Geräten dienen, wie z.B. Toastern oder Waschmaschinen. Die erste Java-Version 1.0 erschien 1996 mit einer kleinen Menge an Standardpaketen, mit welchen es aber schon möglich war Applets[8] zu erstellen. Seinen Namen verdankt Java einer starken Kaffee-Sorte (Java-Bohne), die von den Entwicklern bevorzugt getrunken wurde.

Java zählt zu den objektorientierten Programmiersprachen. Darunter versteht sich ein Programmierparadigma[9], welches auf der Erstellung und Verwendung von Objekten basiert. Mit Hilfe dessen können große Softwareprojekte einfacher verwaltet und der Grad der Wiederverwendbarkeit von verschiedenen Klassen und Funktionen wird wesentlich erhöht werden.[10]

Um Objekte im Zusammenhang mit Paradigmen etwas genauer erklären zu können, werden nun die Prinzipien der *objektorientierten Programmierung* (OOP) grob aufgezeigt:

- *Klasse*
  Die Struktur eines Objektes wird durch die Attribute (auch Eigenschaften) seiner Klassendefinition festgelegt. Das Verhalten des Objektes wird von den Funktionen der Klasse bestimmt.

- *Abstraktion*
  Objekte im System können als ein abstraktes Modell von realen Objekten betrachtet werden, das Funktionen ausführen oder Eigenschaften zurückgeben kann.

- *Datenkapselung*
  Attribute und/oder Funktionen von Objekten können verborgen werden. Somit ist ein direkter Zugriff auf diese nicht möglich und ein unerwartetes oder unbeabsichtigtes Aufrufen einer Funktion wird unterbunden. Der Zugriff erfolgt statt dessen über definierte Schnittstellen.

[8] Java-Applet - kleines Computerprogramm, welches in einem Web-Browser läuft und in der Programmiersprache Java geschrieben ist.
[9] Ein der Programmiersprache oder Programmiertechnik zugrundeliegendes Prinzip
[10] Wikipedia[34]: Objektorientierung, Programmierparadigma, Objekt (Programmierung)

- *Polymorphie*
  Objekte können auf Nachrichten unterschiedlich reagieren. Dies bedeutet z.B., das Funktionen innerhalb eines Objektes „überladen“ werden, sprich mehrmals mit dem gleichen Namen vorkommen können.

- *Vererbung*
  Neue Objekte können aus vorhanden Objekten abgeleitet werden. Somit erbt das abgeleitete Objekt die Attribute und Funktionen des vorhandenen Objektes. Weiterhin können neue Bestandteile hinzugenommen oder vorhandene überladen werden.

Um Java-Anwendungen auf Maschinen (z.B. Computer, Mobiltelefon) ausführen zu können, wird eine *Java Runtime Environment* (JRE - Java-Laufzeitumgebung) benötigt. Sie ist Bestandteil von Java SE und besteht vereinfacht gesagt aus der Java VM und der Java API, enthält aber keine Entwicklungswerkzeuge wie beispielsweise Compiler. Die JRE kann als virtueller Computer betrachtet werden, bei dem die Java VM den Prozessor und die Java API die oberste Bibliotheksschicht darstellt.

Sollen Java-Anwendungen programmiert werden, wird das *Java Development Kit* (JDK) benötigt. Es beinhaltet die JRE und weitere Entwicklungswerkzeuge. Die wichtigsten zwei dürften wohl der Java-Compiler und Java-Doc sein. Der Java-Compiler ist, wie bereits erwähnt, der Übersetzer, der den Quellcode in Maschinencode, Bytecode etc. kompiliert. Mit Hilfe von Java-Doc können Dokumentationen über den Quellcode (Klassen und Schnittstellen) erstellt werden.

3.2.2 Java Virtual Machine

Zusammenfassend lässt sich die JVM als Schnittstelle zwischen Maschine und dem Betriebssystem erklären. Am besten können Funktionsweise, -ablauf und Besonderheiten aber anhand der Abbildung 3.5 beschrieben werden:

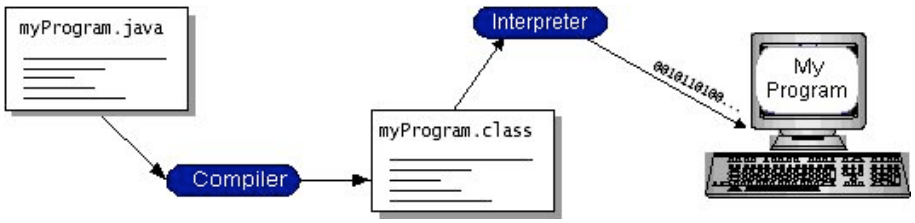


Abbildung 3.5: Java Virtual Machine (The Java Tutorial 2005)

Um eine Java-Anwendung zu erstellen, wird der Quellcode in einer Textdatei mit der Endung .java gespeichert. Diese Quellcode-Datei wird mit Hilfe des Java Compiler (javac)

in eine .class Datei kompiliert. Dieser Vorgang bedeutet, dass der menschenlesbare Programmcode in Maschinencode übersetzt wurde. Allerdings handelt es sich hierbei nicht um einen gewöhnlichen Maschinencode sondern um einen Bytecode, einen sogenannten Zwischencode. Dieser ist in der Regel maschinenunabhängig und im Gegensatz zum Quellcode oft relativ kompakt. Auf Grund seiner Maschinenunabhängigkeit kann er nicht von der Maschine/Prozessor gelesen und verarbeitet werden. Der erstellte Bytecode des Java Compilers wurde für eine virtuelle Maschine übersetzt. Somit kommt die JVM als Software ins Spiel, welche diesen übersetzten Bytecode lesen und vor allem ausführen kann. Das Besondere dabei ist, dass der Programmcode nicht vom Prozessor (Hardware) sondern von der JVM (Software) ausgeführt wird. Weil diese Java Virtual Machine für viele Betriebssysteme (z.B. Windows, Linux, Mac etc.) verfügbar ist, können aus dem vorher erwähnten Grund Java-Anwendungen in der Regel auf diesen unterschiedlichen Systemen incl. Prozessoren laufen.

3.2.3 Java Application Programming Interface

In der Java API werden öffentliche bereits verfügbare Java-Standard-Klassen und Schnittstellen in Sammlungen, so genannten *packages*, zusammengefasst. Dies wird als die Java Klassenbibliothek bezeichnet. Der Zugriff auf diese Bibliothek erfolgt durch die *Java Application Programming Interfaces* (Java API). Häufig wird auch schon die Klassenbibliothek als API bezeichnet. Das bedeutet, dass die Java API bereits eine Vielzahl von verschiedenen Standard-Klassen und Funktionen beinhaltet, die unter Umständen durch deren Nutzung einem Entwickler oder einer Anwendung das Leben sehr stark vereinfachen können. In der ersten Version von Java 1.0 umfasste die Java API rund 212 Klassen verteilt auf 8 packages. In der aktuellen Version 5 sind es mehr als 3200 Klassen und Schnittstellen, welche sich auf über 166 packages verteilen. Ein großer Vorteil dieser API ist die ausgezeichnete Dokumentation. Des Weiteren können Entwickler mit Hilfe von Java-Doc auch selbst solche Dokumentationen erstellen.

Weil der Begriff des *Interfaces* später noch von Bedeutung ist, soll er nun noch etwas ausführlicher beschrieben werden. Mit Hilfe eines *Interfaces* (dt. Schnittstelle) kann für eine Klasse festgelegt werden, welche Funktionen sie unterstützen/anbieten muss und an welchen Stellen (Kontext) die Klassen verwendet werden dürfen. Bei der Erstellung solcher Interfaces muss von dessen Entwickler eine ausführliche Dokumentation erstellt werden, weil sonst der spätere Nutzer (Programmierer) keine Chance hat, diese Schnittstellen zu verwenden.

Für meine Diplomarbeit wurde die JDK 5.0 Update 6 verwendet. Als *Integrated Development Environment* (IDE - dt. Entwicklungsumgebung) wurde Eclipse[6], welches ab

der Versionen 3.1 Java 5.0 unterstützt, genutzt. Eclipse ist ein Open-Source-Framework. Dies bedeutet, dass jeder die Erlaubnis hat, in den Quellcode Einblick zu nehmen, ihn zu verändern und weiterzugeben. Die komplette IDE ist in Java geschrieben, wodurch es auch auf anderen Systemen läuft und kostenlos genutzt werden kann.

Alternative Java-IDEs wären z.B. NetBeans, JBuilder der Firma Borland, JCreator von Xinox Software, IntelliJ IDEA von der Non-Profit-Organisation JetBrains oder Oracles JDeveloper.

3.3 XML

Die *Extensible Markup Language* (dt. erweiterbare Auszeichnungs-Sprache; abgekürzt XML) ist ein Standard zur Darstellung von hierarchischen Datenstrukturen in Dokumenten, der vom *World Wide Web Consortium* (W3C)[32] definiert wird. XML definiert dabei den Aufbau dieser Dokumente durch eine maschinen- und menschenlesbare Baumstruktur. Mit Hilfe von *Document Type Definitions* (DTD - klassische Version) oder *XML-Schemata* (XSD) kann die Struktur von XML-Dokumenten vorgegeben werden. Damit können z.B. Programme über eine XSD festgelegte Schnittstelle Daten und Informationen austauschen. Des Weiteren kann XML für die Konfiguration oder Datenspeicherung von Programmen genutzt werden.

XML-Schema[11]
XML-Schema ist eine komplexe Sprache zur Beschreibung eines XML-Typsystems. Dieses XML-Typsystem umfasst die Spezifikation neuer XML-Elemente, deren Attribute und Kinder-Elemente. Im Gegensatz zu DTD kann bei Verwendung von XML-Schema zwischen dem Namen des XML-Typs und dem in der Instanz verwendeten XML-Tagnamen unterschieden werden.

Geographic Markup Language (GML)[12]
GML ist ein Datenformat zum Austausch raumbezogener Objekte („Features“). GML ist eine Anwendung von XML und durch entsprechende Schemabeschreibungen (XML-Schemadateien *.xsd) festgelegt. GML erlaubt die Übermittlung von Objekten mit Attributen, Relationen und Geometrien im Bereich der Geodaten. GML wird vom Open Geospatial Consortium festgelegt.[19]

[11] aus Wikipedia[34]: XML-Schema
[12] aus Wikipedia: GML

3.4 Route Service

Um den Route Service zu implementieren, sind verschiedene Techniken und Anwendungen notwendig. Da der Route Service dieser Diplomarbeit einen Web Service darstellt, wird zuvor der Begriff des Web Services beschrieben.

3.4.1 Web Service

Web Services haben sich in den letzten Jahren von einer Idee zu einer einsetzbaren Basis-Technologie für die Kommunikation von Software-Systemen entwickelt. Seitdem gibt es verschiedene Definitionen für *Web Service* [12], deswegen soll hier die Definition des W3C als Grundlage der Begriffserklärung verwendet werden:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.[33]

Es handelt sich um Software-Systeme mit klar definierten Schnittstellen zur Kommunikation mit anderen Systemen über ein Netzwerk. Ganz allgemein können Web Services (Web Dienste) auch als Server-basierende Anwendungen bezeichnet werden. Die Schnittstellen werden durch ein XML-Dokument beschrieben. Die *Web Service Description Language* (WSDL) ist standardisiert, ebenso wie das eingesetzte XML-Nachrichtenformat *Simple Object Access Protocol* (SOAP). Das Protokoll zur Kommunikation wird durch den Standard nicht definiert, klassischerweise erfolgt die Kommunikation über HTTP. Mehr Informationen zur Kommunikation mit einem Web Service werden in Kapitel 3.4.3.3 erwähnt.

3.4.2 Verwendete Techniken

Bei der Implementierung des Route Services wurden verschiedene Techniken verwendet. Dieses Unterkapitel soll einen Überblick zu den verwendeten Plattformen und dafür benötigten Techniken geben.

3.4.2.1 Servlet

Das Wort *Servlet* kann nicht vom englischen ins deutsche übersetzt werden. Dieses Wort ist eine Kreation aus „Server“ und „Applet“. Somit wäre eine Übersetzung als *serverseitige Anwendung* möglich. Servlets bilden die Grundlage, um Web Services aufzubauen. Nun zur technischen Beschreibung des Begriffes. Servlets sind Java Klassen, welche in den plattformunabhängigen Bytecode kompiliert und dynamisch von einem Web Server geladen werden können [4]. Dieser Web Server muss als Erweiterung einen *Servlet Container* besitzen, auch *Servlet Engine* genannt, welcher folgende Servlet Funktionalitäten verwaltet: den Lebenszyklus wie starten, stoppen, neu laden und entfernen, Kommunikation mit Clients über das bekannte Request/Response Pair (Abbildung 2.4 auf Seite 13).

Tomcat
Als *Servlet Container* wurde bei dieser Diplomarbeit der Apache Tomcat 5.5.15 von der Apache Software Foundation verwendet. Ein Vorteil des Apache Tomcat ist, dass er nicht nur den notwendigen Servlet Container sondern darüber hinaus noch einen JavaServer Pages Container beinhaltet.[1] Eine sehr angenehme Nebensache war, dass es für die verwendete IDE (Eclipse) ein PlugIn gab, mit welchen der Tomcat Server gestartet, gestoppt oder neu gestartet werden konnte.

3.4.2.2 XMLBeans

Um die XML-kodierten Requests, die an den zu implementierenden Service gehen, verarbeiten zu können, gibt es verschiedene Möglichkeiten. Die zwei wohl bekanntesten XML-Parser[13] sind *Simple API for XML* (SAX) oder *Document Object Model* (DOM). Eine andere Technologie zum Einbinden von XML Dateien in Java ist *XMLBeans*, welches ein Teil des Apache XML Projektes ist. Mit Hilfe von XMLBeans können somit die XML-Requests ausgelesen und verarbeitet werden. Umgekehrt ist es möglich, mit XMLBeans XML-codierte Responses zu erstellen. XMLBeans besteht hauptsächlich aus drei APIs:

1. *XMLObject*
   Es wird eine XML-Schema Definition verwendet, um Klassen und Schnittstellen zu kompilieren. Daraus entstehen dann Java Objekte, die wiederum - wie bei den Java Beans geläufig - mittels Getter- und Setter-Methoden bearbeitet werden können.

[13] aus Wikipedia[34]: Ein Parser (engl. to parse „analysieren“ bzw. von lateinisch pars „Teil“; weshalb Parser im Deutschen gelegentlich auch als Zerteiler bezeichnet werden) ist ein Computerprogramm/-Programmteil, das entscheidet, ob ein Eingabetext zur formalen Sprache einer bestimmten Grammatik gehört.

2. *XMLCursor*
   Von jedem möglichem XMLObject kann ein XMLCursor erstellt werden.
   Dieser Cursor kann durch den gesamten XML Baum bewegt werden.

3. *SchemaType*
   XMLBeans liefert ein volles XML Document Object Model - (XML DOM).

3.4.2.3 GeoTools

Was vor Jahren noch undenkbar war, ist heute schon zur Normalität geworden. Quellcode, Klassen, Methoden und Werkzeuge für Geoinformationssysteme[14] sind im *World Wide Web* (WWW) „frei erhältlich“[15]. Für diese Diplomarbeit soll die Bibliothek vorgestellt werden, aus welcher verschiedene Klassen und Funktionen genutzt und teilweise auch geändert wurden.

Das Projekt GeoTools gibt es bereits seit 1996. Inzwischen besteht es aus zwei verschiedenen Versionen, GeoTools 1.0 oder auch GeoTools-Lite und GeoTools 2 (GT1 und GT2). GT1 wurde von 1996-2003 entwickelt. Seit 2002 wird an GT2 gearbeitet, welches sich noch in Unterversionen gliedert (2.0,2.1,2.2), die aber hier nicht weiter beschrieben werden sollen. Wenn in dieser Diplomarbeit von GeoTools die Rede ist, ist damit immer GeoTools 2 (GT2) gemeint.

GeoTools ist ein Open-Source-Code Java GIS Toolkit und bietet eine Sammlung von verschiedenen Klassen und Funktionen, mit deren Hilfe es möglich ist, Geodaten u.a. zu reorganisieren, analysieren und zu speichern. Ein GeoTools Release besteht in der Regel aus drei Dingen:

- Sammlung aller Bibliotheken (JAR-Dateien[16])
- Sammlung aller Quellcode Dateien
- einer JavaDoc[17]

Dabei hat GeoTools eine modulare Architektur, welches es auf einfache Weise erlaubt, zusätzliche Funktionalität hinzuzufügen oder zu entfernen. Das Ziel des GeoTools Projektes ist, eine Sammlung von Java Objekten in einem Framework zu entwickeln, mit

[14] Geographisches Informationssystem (GIS) ist ein Informationssystem, mit dem „raumbezogene Daten digital erfasst und redigiert, gespeichert und reorganisiert, modelliert und analysiert sowie alphanumerisch und graphisch präsentiert werden.“[2]
[15] Mehr Informationen unter: http://freegis.org
[16] JAR - Java ARchive
[17] Javadoc - HTML-Dokumentationsdateien über den Java-Quellcode

deren Hilfe es möglich ist, OGC-konforme Dienste zu implementieren oder OGC Kompabilität in Anwendungen oder Applets zur Verfügung zu stellen. GT2 enthält dafür eine Kern API mit Interfaces und Standard Implementierungen von diesen Interfaces.[10]

GeoTools ist unter der GNU *Lesser General Public License* (LGPL) lizenziert. Was unter dieser Lizenz zu verstehen ist, wird in Kapitel 3.6.2 beschrieben.

3.4.2.4 Java Topology Suite (JTS)

Die *Java Topology Suite* (JTS)[14] ist eine vollständig in Java geschriebene API für geometrische Operationen und Funktionen im 2D Raum. Damit bietet diese API sehr gute Grundlagen für die Entwicklung von Anwendungen, welche mit räumlichen Datensätzen zu tun haben. Das Ziel von JTS ist es, eine komplette, konsistente und robuste Implementierung von räumlichen 2D Algorithmen anzubieten. Dabei sollte sie zusätzlich konform zu der OGC *Simple Features Specification for SQL*[20] sein. Hinzu kommt, dass sie, wie GeoTools, unter der LGPL lizenziert ist.

Eine gute Übersicht, eines kleinen Teils der verwendeten Techniken/Programme, bietet die folgende Abbildung.

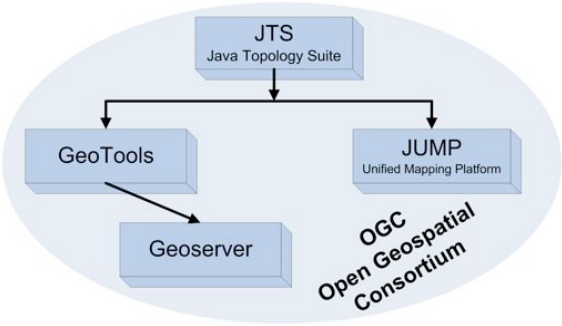


Abbildung 3.6: Überblick Techniken/Programme im Zusammenhang mit OGC

3.4.2.5 PostgreSQL / PostGIS

Um einem Routenplaner die Daten, die er benötigt, zur Verfügung zu stellen, gibt es mehrere Möglichkeiten. Eine besteht darin, die erforderlichen Daten (also z.B. Straßennetz oder Adressverzeichnis) in einer Datei zu hinterlegen. Eine andere, bessere Variante ist, die Daten in einem Datenbanksystem abzulegen. Dies würde verschiedene Vorteile mit sich bringen. Bevor jetzt Einzelheiten über das ausgewählte System erläutert werden, soll zunächst kurz der Begriff *Datenbanksystem* (DBS) beschrieben werden. Ein

DBS ist eine Kombination aus zwei Teilen, *Datenbank* (DB) und *Datenbankmanagementsystem* (DBMS). Eine DB ist eine elektronische, systematische Sammlung von zusammenhängenden (in Beziehung zueinander stehenden) Daten in einem Datenspeicher. Das DBMS hat die Aufgabe, Daten in der DB zu speichern, zu organisieren, zu modifizieren oder auszulesen. Dies alles geschieht über eine sogenannte Datenbanksprache. Mit ihrer Hilfe können Daten u.a. eingefügt, abgefragt, geändert und gelöscht werden. Das bekannteste Beispiel für solch eine Sprache ist SQL[18].

PostgreSQL ist ein *objektrelationales Datenbankmanagementsystem* (ORDBMS), das als Open-Source-Programm frei verfügbar ist und ohne Lizensierung benutzt werden darf. Ursprünglich wurde Postgres als universitäres Projekt an der University of California at Berkeley Computer Science Department entwickelt. Seither wurde von vielen Entwicklern auf der ganzen Welt an diesem Code weitergearbeitet und erhielt 1996 den Namen PostgreSQL.[28][27]

Das Besondere an PostgreSQL ist eine Erweiterung Namens PostGIS[26], welche für PostgreSQL eine Unterstützung von geografischen Objekten bietet. Damit ist es möglich, durch PostGIS einen „spatially“ (dt. räumlichen) PostgreSQL Server zu erstellen, welcher auch für Geoinformationssysteme genutzt werden kann. Bei der Speicherung der Daten hält sich PostGIS an die OGC Spezifikation *Simple Features Specification for SQL*[20]. Diese Simple Features sind z.B. Punkt, Linie und Polygon. Ein weiterer großer Vorteil von PostGIS ist, dass damit zahlreiche räumliche Analyse- und Abfragefunktionen zur Verfügung gestellt werden können. PostGIS ist von Refractions Research[19] innerhalb eines Forschungsprojektes als eine Open-Source räumliche Datenbank-Technologie entwickelt und unter der GNU *General Public License* (GPL) veröffentlicht worden.

JDBC

Der Zugriff auf eine Datenbank erfolgt mit Hilfe der *Java Database Connectivity* (JDBC), die dafür eine geeignete Schnittstelle bietet, JDBC API[13]. Die Aufgaben der JDBC sind folgende:

- Verbindungen zur Datenbank aufbauen, sie verwalten und wenn sie nicht mehr benötigt werden, abzubauen
- SQL-Anfragen an die Datenbank weiterleiten, die zurückkommenden Antworten in eine Java nutzbare Form umwandeln und dem Programm zur Verfügung stellen

[18] Mehr Informationen z.B. unter Wikipedia[34]: SQL
[19] Mehr Informationen unter: http://www.refractions.net

3.4.3 Geoserver

Damit der Route Service die berechnete Route auch visuell z.B. mittels einer Karte mit eingezeichneter Route darstellen kann, wird ein Web Service benutzt, welcher die Erstellung der Karte übernimmt. In Wirklichkeit gehört aber viel mehr dazu, als nur *„ein“* Web Service.

Kurz zur Geschichte von Geoserver[9]: GeoServer wurde durch *The Open Planning Project* (TOPP), eine aus New York stammende Non-Profit Gemeinschaft, gestartet. TOPP hatte sich zur Aufgabe gemacht, behördliche und räumliche Planungsentscheidungen für die Öffentlichkeit und vor allen den Betroffenen transparenter zu gestalten. 2003 kam die erste Geoserver Version als ein Teil des OGC CITE[20] Projektes zu Stande, die Geoserver Version 1.0.[21]

Geoserver ist ein Open-Source Server, welcher die Funktionalität eines *Web Feature Service Transactional* (WFS-T) und eines *Web Map Service* (WMS) mit *Styled Layer Descriptor* (SLD) Unterstützung gemäß der OGC Spezifikationen bietet (beim WMS die Version 1.1.1 und beim WFS die Version 1.0). Damit sind folgende Operationen durch einen Geoserver für einen Benutzer möglich:

- erstellen von Karten/Bildern (WMS),
- abrufen von aktuellen geographischen Daten (WFS) und
- der Benutzer kann diese vorhandenen Daten aktualisieren, löschen oder neue hinzufügen (WFS-T).

Geoserver baut auf GeoTools auf und hilft bei deren Weiterentwicklung mit. Besonders erwähnenswert ist die einfache Art und Weise mit welcher ein Geoserver installiert und konfiguriert werden kann. Auch die Daten lassen sich ohne größeren Aufwand integrieren. Alles geschieht sehr komfortabel über eine HTML[22]/JSP[23]-Benutzerschnittstelle, welche mit einem Web-Browser genutzt werden kann.

Für die Implementierung des Route Services wurde die Geoserver Version 1.3.0[9] verwendet. Auf den Hintergrund, warum Geoserver und nicht ein anderer WFS/WMS, wie z.B. ein UMN MapServer[24] verwendet wurde, wird nach den Erklärungen des WMS, WFS-T und von SLD eingegangen.

[20] CITE - Compliance & Interoperability Testing & Evaluation Initiative ;
Online unter: http://cite.opengeospatial.org
[21] Mehr Informationen unter: http://docs.codehaus.org/display/GEOSDOC/Introduction
[22] HTML - Hypertext Markup Language
[23] JSP - JavaServer Pages
[24] Mehr Informationen unter: http://ms.gis.umn.edu

3.4.3.1 WMS

Von Geoserver wird die Version 1.1.1 der WMS Spezifikation verwendet:

> A Web Map Service (WMS) produces maps of georeferenced data. We define a „map“ as a visual representation of geodata; a map is not the data itself. This specification defines three WMS operations: GetCapabilities returns service-level metadata, which is a description of the service's information content and acceptable request parameters; GetMap returns a map image whose geospatial and dimensional parameters are welldefined; GetFeatureInfo (optional) returns information about particular features shown on a map. This specification defines a syntax for World Wide Web (WWW) Uniform Resource Locators (URLs) that invoke each of these operations. Also, an Extensible Markup Language (XML) encoding is defined for service-level metadata.[24]

Ein WMS ist also ein Web Service, welcher aus georeferenzierten Daten eine Karte erstellt. Diese Karte wird vom OGC dabei als eine visuelle Repräsentation von Geodaten definiert. Die erstellte Karte wird in Form eines Bildes in einem vorgebbaren Dateiformat vom WMS auf eine Anfrage zurückgegeben. Die Auswahl der Dateiformate ist jeweils abhängig von der Implementierung des WMS, möglich wären z.B. JPEG, PNG, SVG etc.. Ein WMS kann laut der Spezifikation drei verschiedene Operationen unterstützen, zwei davon muss er anbieten (required - dt. erforderlich):

1. *GetCapabilities* (erforderlich) - Mit dieser Operation wird nach den Fähigkeiten des WMS gefragt. Als Antwort erfolgt ein XML-Dokument mit Metainformationen, welches u.a. Angaben zum Anbieter, die unterstützten Ausgabeformate und abfragbaren Layer des WMS beinhaltet.

2. *GetMap* (erforderlich) - Diese Anfrage liefert die Karte zurück.

3. *GetFeatureInfo* (optional) - Die Operation kann optional von einem WMS angeboten werden. Ein WMS kann freiwillig Anfragen zu dem dargestellten Kartenausschnitt beantworten. Als Antwort könnte er also thematische Informationen über die angefragten Daten zurückgeben.

Um diese Operationen eines WMSs nutzen zu können, erfolgt mit ihm eine Kommunikation über das *Hypertext Transfer Protocol* (HTTP). Da ein WFS über die selbe Art & Weise die Kommunikation pflegt, soll sie nach der Erklärung des WFSs erfolgen. Von den eben erwähnten drei Operationen wird nur die GetMap bei der Implementierung des Route Service benötigt.

GetMap-Operation

Wie eben bereits erwähnt, liefert die Anfrage an den WMS eine Karte. Bei dieser Anfrage besteht jedoch die Möglichkeit, durch verschiedene Parameter die Karte an die eigenen Bedürfnisse anzupassen. Die folgende Tabelle bietet hierfür eine Übersicht:

Tabelle 3.1: WMS GetMap - Parameter

Request	Parameter	E/O	Beschreibung
VERSION	WMS-Version	E	Version des WMS (z.B. *1.1.1*)
REQUEST	GetMap	E	Anfrage-Operation
LAYERS	Layer-Liste	E	Kommagetrennte Liste der Layer
STYLES	Style-Liste	E	Kommegetrennte Liste der Styles
SRS	Namensraum:Bezeichner	E	Koordinatensysten(z.B. *EPSG:4326*)
BBOX	minX,minY,maxX,maxY	E	Ecke links unten & rechts oben der BBox[*1]
WIDTH	Ausgabe-Breite	E	Breite der Karte
HEIGHT	Ausgabe-Höhe	E	Höhe der Karte
FORMAT	Ausgabe-Format	E	Ausgabeformat Karte
TRANSPARENT	true/false	O	Hintergrundtransparenz der Karte (Standard *false*)
BGCOLOR	Farbwert	O	Hintergrundfarbe der Karte in Hexadezimalschreibweise
EXCEPTIONS	Ausgabeformat	O	Format, in dem Fehler des WMS zurückgegeben werden
TIME	Zeit	O	Datum des angefragten Layers
ELEVATION	Höhenangabe	O	Höhe des angefragten Layers
SLD[*2]	SLD-URL	O	URL des zu verwendenden SLD
WFS[*2]	WFS-URL	O	URL zu einem WFS, der Features beinhaltet, welche mit SLD symbolisiert werden sollen
E = Erforderlich(Required) ; O = Optional			

[*1] BoundingBox - Rechteck, welches durch die minimalen & maximalen Koordinaten, z.B. einer Linie, aufgezeigt wird.

[*2] Diese Operationen gelten nur für WMSs, die die SLD Spezifikation unterstützen.

3.4.3.2 WFS/-T

Ein *Web Feature Service* (WFS) ist ein Dienst, welcher einen internetgestützten Zugriff auf Geodaten ermöglicht, welche in Datenbanken, einem Dateisystem (z.B. ESRI Shapefile) oder ggf. in einem entfernten WFS liegen. Der Zugriff erfolgt dabei gemäß der WFS

Spezifikation des OGC (Geoserver nutzt WFS Spezifikation Version 1.0[23]). Die Bedeutung von Geodaten wird in Kapitel 3.5 erklärt. Beim Anfordern von Geodaten können des Weiteren standardisierte XML-Filter-Aktionen (OGC *Filter Encoding Implementation Spezifikation*[21]) zur Findung der gesuchten Geodaten beitragen. Ein Beispiel für solch einen Filter wäre: *„Suche alle Straßen, die Autobahnen sind!“*. Die Standard-Antwort eines WFS erfolgt in der *Geographic Markup Language* (GML). Bei einem WFS muss zwischen zwei verschieden Typen unterschieden werden. In einer „einfachen“ Version bietet der WFS nur lesenden Zugriff auf die Daten mit folgenden drei Operationen:

1. *GetCapabilities* (erforderlich) - Mit dieser Operation wird nach den Fähigkeiten des WFS gefragt. Als Antwort erfolgt ein XML-Dokument mit Angaben zum Anbieter des WFS, abfragbaren Feature Types und möglichen Operationen.

2. *DescribeFeatureType* (erforderlich) - Ist eine Anfrage, bei welcher Informationen zur Struktur der einzelnen Feature Types zurückgegeben werden.

3. *GetFeature* (erforderlich) - Beinhaltet eine Anfrage, bei welcher die eigentlichen (gefilterten) GeoDaten zurückgegeben werden.

Eine erweiterte Version ist ein *WFS-Transactional* (WFS-T). Dies bedeutet, ein transaktionaler WFS bietet die Operationen eines „einfachen“ WFS und darüber hinaus auch den schreibenden Zugriff auf die Daten mit folgenden Operationen:

1. *Transaction* (erforderlich) - Diese Operation stellt die Möglichkeiten Insert, Update und Delete bereit. Damit können Features in der Datenbasis eingefügt, aktualisiert oder gelöscht werden.

2. *LockFeature* (optional) - Bietet eine Option, dass bei einer Transaction das Feature nicht von einer anderen Operation geändert werden kann.

Die einzelnen Parameter der Operationen sollen hier nicht weiter erwähnt werden, weil sie für diese Diplomarbeit nicht von Bedeutung sind. WMS und WFS müssen miteinander verbunden sein, damit ein WMS Karten aus Geodaten erstellen kann. Normalerweise erfolgt diese Kommunikation auf Basis der OGC Schnittstellen-Spezifikationen. Beim Geoserver allerdings funktioniert dieses durch eine eigene, interne Schnittstelle zwischen den beiden Services.

3.4.3.3 Protokoll für den Datenaustausch

Anfragen an die Services werden mit Hilfe des *Hypertext Transfer Protocols* (HTTP)[11] durchgeführt. Die Funktionsweise lässt sich gut an der Abbildung 3.7 zeigen.

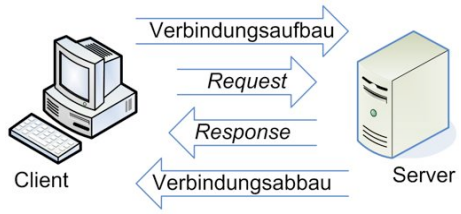


Abbildung 3.7: Funktionsweise des HTTP für den Datenaustausch

HTTP 1.1 unterstützt acht Anfrage (Request)-Methoden, davon sind aber nur zwei für die hier erwähnten Web Services wichtig, *GET* und *POST*. Eine oder beide Methoden müssen von einem OGC Web Service angeboten werden.

GET-Methode

Die Get-Methode besteht aus der URL[25] des Web Services und zusätzlichen Parametern, welche mit verschiedenen Trenn- und Verbindungszeichen angehängt werden. Das Trennzeichen ist z.B. das "?". Es steht für das Ende der URL und den Beginn der Parameter. Durch ein "&" werden die verschieden zu übertragenen Parameter getrennt. Sollen für einen Parameter mehrere Werte übertragen werden, müssen diese durch ein "," getrennt werden. Zu erwähnen ist noch, dass die Reihenfolge der Parameter, sowie deren Schreibweise (ob groß oder klein) keine Rolle spielt. Hingegen muss aber auf die genaue Schreibweise der Werte, die an einem Parameter „hängen", geachtet werden. Ein Nachteil von GET ist, dass maximal 255 Zeichen mit der URL übertragen werden können. Im Folgenden ist eine GET-Anfrage an den WMS vom Geoserver dargestellt:

```
1 http://localhost:8080/geoserver/wms?REQUEST=GetMap
2  &BBOX=3432965.30,5795979.46,3434824.66,5797828.83
3  &SRS=EPSG:31467&LAYERS=topp:strassen-joined&STYLES=strassen
4  &FORMAT=image/png&WIDTH=400&HEIGHTt=400&TRANSPARENT=false&BGCOLOR=0xFFFFFF
```

Listing 3.1: GET-Methode - GetMap

Zum besseren Verständnis wird der Request jetzt gesplittet:
URL: *http://localhost:8080/geoserver/wms*
Parameter 1: *REQUEST* ; Wert 1= *GetMap*
Parameter 2: *BBOX* ; Wert 2.1 = *3432965.30* , Wert 2.2 = *5795979.46* , . . .
. . .

[25] Uniform Resource Locator (URL) - umgangsprachlich „Web-Adresse" oder nur „Adresse"

POST-Methode

Die POST-Methode kann nicht ganz so einfach mit Hilfe eines Web-Browsers genutzt werden wie die GET-Methode. Bei der POST-Methode werden, vereinfacht gesagt, die Parameter im Body des Requests gesendet und sind damit nicht sichtbar. Um die POST-Methode in einem Web-Browser nutzen zu können, wird eine Website benötigt, bei welcher Daten in ein Formular-Feld geschrieben und darüber gesendet werden können.

Dieselbe Anfrage, welche mit Hilfe der GET-Methode an den WMS eine Karte geliefert hat, ist auch mit der POST-Methode möglich (Ausnahme Geoserver, nicht jeder WMS kann POST entgegennehmen). Beim Geoserver ist eine Website verfügbar, in welche Daten (XML-Dokument) eingetragen und an den WMS gesendet werden können.

Wird also an folgende URL: http://localhost:8080/geoserver/wms/GetMap
das nachfolgende XML-Dokument gesendet, antwortet der WMS mit der gleichen Karte wie bei der GET-Methode verwendeten URL incl. deren Parameter.

```
1  <GetMap version="1.2.0" service="WMS" xsi:schemaLocation="http://www.opengis.net/ows
2   C:/Programme/xampp/tomcat/webapps/geoserver/schemas/sld/GetMap.xsd"
3  xmlns="http://www.opengis.net/ows"
4  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5  xmlns:sld="http://www.opengis.net/sld" xmlns:gml="http://www.opengis.net/gml">
6    <sld:StyledLayerDescriptor version="1.0.0">
7      <sld:NamedLayer>
8        <sld:Name>topp:strassen-joined</sld:Name>
9        <sld:NamedStyle><sld:Name>strassen</sld:Name></sld:NamedStyle>
10     </sld:NamedLayer>
11   </sld:StyledLayerDescriptor>
12   <BoundingBox srsName="31467">
13     <gml:coord><gml:X>3432965.30</gml:X><gml:Y>5795979.46</gml:Y></gml:coord>
14     <gml:coord><gml:X>3434824.66</gml:X><gml:Y>5797828.83</gml:Y></gml:coord>
15   </BoundingBox>
16   <Output>
17     <Format>image/png</Format>
18     <Transparent>false</Transparent>
19     <BGcolor>#FFFFFF</BGcolor>
20     <Size><Width>400</Width><Height>400</Height></Size>
21   </Output>
22 </GetMap>
```

Listing 3.2: POST-Methode - GetMap XML-Dokument

Wird das verwendete XML-Dokument etwas genauer betrachtet (Zeilen 6-11), wird das Kürzel SLD - *Styled Layer Descriptor* - sichtbar. Was alles hinter und mit SLD im Zusammenhang steht, wird als nächstes beschrieben.

3.4.3.4 SLD

SLD wurde im Verlaufe meiner Diplomarbeit zu einem sehr wichtigen Bestandteil.

Durch die *Styled Layer Descriptor* (SLD)[22] Spezifikation werden die begrenzten Möglichkeiten eines WMSs bzgl. der Kartendarstellung erweitert. Ein normaler WMS bietet seine Daten in Layern an. Karten werden aus diesen Layern und diversen vordefinierten Stylen erstellt. Dies bedeutet, es kann nur ein Style für ein gesamtes Layer verwendet werden. Mit Hilfe von SLD ist eine sogenannte User-definierte Symbolisierung von Karten möglich. Daten können also durch SLD klassifiziert werden, was soviel heißt wie, Daten können anhand ihrer Attribute in verschiedenen Klassen sortiert/gefiltert werden. Die einzelnen Klassen können dann durch selbst vorgegebene Styles dargestellt werden.

Zum besseren Verständnis nachfolgend zwei Beispiele:

(a) Standard Karte (b) Karte mit SLD gestyled

Abbildung 3.8: Karte von WMS ohne und mit SLD gestyled

An den Karten in Abbildung 3.8 ist sehr gut der Unterschied zu erkennen, welchen enormen Vorteil SLD mit sich bringt. Bei der Standard Karte (a) wurden jeweils vordefinierte Styles verwendet. Dazu gehören *Default_Line* für Gewässerlinien und Straßen, *Default_Polygon* für Gewässerflächen und Grünflächen und *Default_Point* für POIs.

Bei der rechten Abbildung (b) wurden selbstdefinierte Styles (SLD) verwendet. Damit sind z.B. Grünflächen und Gewässerflächen sehr schön sichtbar zu machen. Ein Teil der Möglichkeiten von SLD wird durch deren Verwendung beim Straßen-Layer sichtbar. Hier wurde das Layer anhand des Straßen-Typs klassifiziert. Bundesstraßen werden gelb und Haupt- oder Nebenstraßen grau gezeichnet, wobei aber Hauptstraßen eine andere (dickere) Linienstärke als Nebenstraßen erhalten.

Es gibt drei Möglichkeiten laut der SLD Spezifikation, um SLD mit einem Client nutzen zu können:

1. Der Client kommuniziert mit dem WMS über die GET-Methode, wobei der Parameter "SLD" (Zeile 3) die URL zu einer SLD-Datei enthält. Dies bedeutet hier, dass die SLD-Datei irgendwo entfernt liegen kann.

```
1 http://yourfavoritesite.com/WMS?
2  REQUEST=GetMap&BBOX=0.0,0.0,1.0,1.0&
3  SLD=http%3A%2F%2Fmyclientsite.com%2FmySLD.xml&
4  WIDTH=400&HEIGHT=400&FORMAT=PNG
```

Listing 3.3: SLD - GET-Methode mit externer SLD Datei

2. Der Client kommuniziert ebenfalls, wie bei der vorherigen Variante, über die GET-Methode mit dem WMS. Allerdings wird die SLD-Datei im GET Request mit Hilfe des Parameters "SLD_BODY" (Zeile 3) untergebracht.

```
1 http://yourfavoritesite.com/WMS?
2  REQUEST=GetMap&BBOX=0.0,0.0,1.0,1.0&
3  SLD_BODY=%3C%3Fxml+version%3D%221.0%22+encoding%3D%22UTF-8%22%3F%3E%3C
4         StyledLayerDescriptor+version%3D%221.1.0%22%3E%3CNamedLayer%3E%3C
5         Name%3ERivers%3C%2FName%3E%3CNamedStyle%3E%3CName%3ECenterLine%3C%2F
6         Name%3E%3C%2FNamedStyle%3E%3C%2FNamedLayer%3E%3CNamedLayer%3E%3C
7         Name%3ERoads%3C%2FName%3E%3CNamedStyle%3E%3CName%3ECenterLine%3C%2F
8         Name%3E%3C%2FNamedStyle%3E%3C%2FNamedLayer%3E%3CNamedLayer%3E%3C
9         Name%3EHouses%3C%2FName%3E%3CNamedStyle%3E%3CName%3EOutline%3C%2F
10        Name%3E%3C%2FNamedStyle%3E%3C%2F
11        NamedLayer%3E%3C%2FStyledLayerDescriptor%3E
12 WIDTH=400&HEIGHT=400&FORMAT=PNG
```

Listing 3.4: SLD - GET-Methode mit SLD Datei im GET Request

3. Der Client kommuniziert mit dem WMS über die HTTP POST-Methode. Dies bedeutet, SLD wird in das XML-Dokument des GetMap Requests eingebettet. Gut zu sehen im Listing 3.2 in den Zeilen 6 bis 11. Des Weiteren fällt auf, dass die Parameter LAYERS und STYLES, durch NamedLayer/Name und NamedLayer/NamedStyle im SLD kodiert und damit ersetzt werden.

Zu erwähnen ist, dass die Varianten eins und zwei nicht mit dem Geoserver getestet wurden. Es kann also keine Aussage darüber getroffen werden, ob sie funktionieren. Sie sollen hier nur als alternative Möglichkeiten aus der SLD Spezifikation zu der Variante drei erwähnt werden. In der Route Service Implementierung wird ausschließlich die letzte Variante genutzt.

Hier muss noch eine weitere Möglichkeit erwähnt werden, mit welcher SLDs in einem WMS genutzt werden können (falls der genutzte WMS dies unterstützt!). Beim Geoserver funktioniert dies folgendermaßen. Es können beliebige SLD Dateien auf den Server hochgeladen werden, welche später einfach über den "STYLES"-Parameter im GetMap Request genutzt werden können. D.h., wird des Öfteren ein und dieselbe SLD-Datei bei einem Request an einen WMS übergeben, kann sie aus verschiedenen Gründen (z.B. GetMap Requests übersichtlicher gestalten) auf den Server gespeichert und anschließend von dort vom WMS genutzt werden. Beispiel hierfür, das Listing 3.1 ; Zeile 3 ; "STYLES=*strassen*". Hier wurde bereits eine SLD-Datei auf den Server gespeichert und durch den Namen „*strassen*" genutzt.

Für das bessere Verständnis werden noch einige Elemente von SLD kurz erklärt. Es sind bei weitem nicht alle, aber die für diese Diplomarbeit wohl Wichtigsten:

- *NamedLayer & UserLayer*
  Ein „Layer" wird als Sammlung von Features definiert. Ein NamedLayer Element wird verwendet, wenn das gewünschte Layer für den WMS verfügbar ist.
  Dagegen können mit Hilfe des UserLayer Elementes Layer im SLD mitgegeben werden, welche nicht für einen WMS verfügbar sind.

- *NamedStyle & UserStyle*
  Von der gleichen Bedeutung „Named" und „User" betreffend, lassen sich die folgenden beiden Begriffe erklären. NamedStyle wurde bereits kurz beschrieben als SLD Dateien, welche auf den Server hochgeladen wurden und durch einen Style-Namen verfügbar sind. Mit einem UserLayer besteht die Möglichkeit einen eigenen definierten Style, welcher nicht auf dem Server verfügbar ist, im SLD mitzugeben.

- *Rule*
  Mit Hilfe des Rule-Elementes werden Regeln für die Klassifizierung und Symbolisierung der Features anhand ihrer Attribute festgesetzt.

- *Filter*
  Durch das Filter-Element kann eine Selektion von Features innerhalb des Rule-Elementes anhand von verschiedenen Attributen erfolgen. Zu erwähnen ist noch, dass die Syntax der Filter-Elemente von der OGC *Filter Encoding Spezifikation*[21] stammen. Folgende drei Beispiele benennen Filter, *PropertyIsEqualTo*, *PropertyIsLessThan*, *PropertyIsGreaterThan*. Diese drei Filter könnten eingesetzt werden, wenn Features gesucht werden, welche den gleichen ("="), einen kleineren ("<") oder einen größeren (">") Attributwert haben.

- *PointSymbolizer, LineSymbolizer, PolygonSymbolizer & TextSymbolizer*
  Mit Hilfe dieser vier Symbolizer können Punkte, Linien, Polygone und Texte den gewünschten Darstellungsanforderungen angepasst werden. Z.B. können Eigenschaften wie Farben, Strichstärken, Farbfüllung, Schriftart, Schriftgröße, Transparenz und vieles mehr geändert werden.

3.4.3.5 Warum Geoserver?

Der Grund, warum Geoserver und nicht z.B. ein *University of Minnesota* (UMN) MapServer verwendet wurde, war folgendes Problem. *Wie kann die ermittelte Route in eine vorhandene Karte eingezeichnet werden?* Zur Lösung dieses Problems kam anfangs die Idee, die ermittelte Route mit Hilfe eines WFS-T und der *insert*-Operation auf dem Server abzuspeichern und anschließend mit Hilfe des WMS eine Karte zu erstellen, in welcher „einfach“ das Layer, in welchem die Route gespeichert wurde, über alle anderen Layer gelegt werden sollte. So zumindest die Theorie zu Beginn dieser Diplomarbeit.

Die Funktionalität eines WFS-T bieten nur sehr wenige an und damit fiel die Wahl auf den Geoserver. Die konkrete Implementierung im Route Service sieht aber wieder komplett anders aus. Während der Einarbeitung in den Geoserver, WMS, WFS/-T und SLD kam es zu einer anderen Lösung. Es hat sich herausgestellt, dass es - vereinfacht gesagt - mit Hilfe von SLD möglich ist, bei einer GetMap-Anfrage an einen WMS ein Userlayer mitzugeben, welches die Geometrien (Linien) der ermittelten Route beinhaltet. Somit war das oben genannte Problem ganz anders gelöst worden. Durch den beim Geoserver integrierten SLD-fähigen WMS sind noch weitere sehr nützliche Sachen möglich, die aber später bei der Implementierung des Route Services weiter beschrieben werden.

3.5 Daten

Um den Route Service zu implementieren und testen zu können, waren verschiedene Testdaten von sehr hoher Bedeutung. Im WWW finden sich hierfür einige Anbieter, die kostenlos Daten für Routenplaner zur Verfügung stellen. Mit diesen Daten lassen sich natürlich nicht Routenplaner für z.B. ganz Deutschland oder Europa anfertigen. Sie können höchstens für eine Stadt genutzt werden. Aber sie sind vollkommen ausreichend, um die Funktionsweise des implementierten Route Services zu testen und aufzuzeigen. Die Wahl der Daten fiel dabei auf die *Freien Vektor-Geodaten Osnabrück* (Frida)[8]. Die Daten wurden anhand von Orthophotos[26] der Stadt Osnabrück digitalisiert. Bei der Di-

[26] Ein Orthophoto ist ein Luftbild der Erdoberfläche, welches verzerrungsfrei und maßstabsgetreu ist.

gitalisierung wurden auf der Basis der Orthophotos geometrische Figuren erstellt. Sprich aus Rasterdaten (Foto) wurden Vektordaten. Dieses wären z.B. Linien für Straßen oder Flüsse, Polygone für Wälder oder Wiesen und Punkte für Sehenswürdigkeiten. Aus den Vektordaten und Attributen wurden anschließend verschiedene Layer der Stadt Osnabrück erstellt. Eine Vektordate, z.B. eine Linie und ihre Attribute (z.B. Straßenname), ist ein Feature. Sammlungen solcher Features werden als Geodaten bezeichnet. Im Folgenden sollen die verschiedenen Layer der Stadt Osnabrück vorgestellt werden:

1. ***Straßennetz***
   Das wichtigste für einen Routenplaner ist das Straßennetz. Hierbei wurde, soweit möglich, bei der Digitalisierung der Daten besonderer Wert gelegt. Das Layer enthält außer den Geometrien der Straßen weitere Attribute:
   1. StraßenID / StraßenName
   2. StraßenTyp: z.B. Autobahn, Bundes-, Haupt-, Neben- und sonstige Straßen.
   3. Spuren: Anzahl der Straßenspuren
   4. Ebene: Anzahl der Straßenebenen

2. ***Fließgewässer***
   Fluss, Bäche mit folgendem Attribut:
   1. FließgewässerID / FließgewässerName

3. ***Gewässerflächen***
   Seen mit folgendem Attribut:
   1. GewässerflächenID / GewässerflächenName

4. ***Grünflächen***
   Grün- und Waldflächen mit folgenden Attributen:
   1. GrünflächenID / GrünflächenName
   2. GrünflächenTyp: z.B. Wald, Grünfläche

5. ***Sehenswürdigkeiten (POIs - Points of Interest)***
   Es wurden verschieden POIs digitalisiert: Parkplätze, Friedhöfe, Kirchen und sonstige besondere Punkte mit folgenden Attributen:
   1. POIName
   2. POITypID / POITypName: z.B. Kirche, Parkplatz, Friedhof

Für die später erwähnte prototypische Implementierung des Route Services mussten die Daten noch verschiedentlich angepasst werden. Die genauen Modifikationen und Probleme (z.B. bzgl. Vollständigkeit, Topologie etc.) werden aber bei der Implementierung des Route Service im Kapitel 4 und den Installationshinweisen im Kapitel 6 behandelt.

3.6 Sonstiges

Im letzten Abschnitt des Kapitels werden folgende Inhalte dargestellt. OpenJUMP als Programm zur Bearbeitung und Erstellung von Daten für den Route Service, eine kurze Erklärung der GNU-LGPL und deren damit verbundenen Folgen und als letzter Punkt EPSG.

3.6.1 OpenJUMP

Um die unter Kapitel 3.5 beschriebenen Daten nachzubearbeiten, zu ergänzen oder um neue Daten zu erstellen, wurde das Programm OpenJUMP verwendet.

JUMP
Die *JUMP Unified Mapping Platform* (JUMP), ist eine GUI[27]-basierende Anwendung für die Betrachtung und Bearbeitung von räumlichen Daten. Sie wurde ursprünglich von Vivid Solutions[31] entwickelt. Es bietet viele Funktionen, welche auch von gängigen GIS-Produkten für die Analyse und die Handhabung von geo-räumlichen Daten bekannt sind.

JUMP Pilot Project (JPP)
Das *JUMP Pilot Project* (JPP) ist eine Organisation, welche sich mit der Entwicklung von Software-Lösungen beschäftigt, die auf JUMP basieren. Sie bietet Entwicklern, rund um die Welt, die Gelegenheit, sich auszutauschen und so doppelte Arbeiten zu vermeiden. Die Haupt-Software-Plattform, die z.Z. durch das JPP koordiniert wird, ist OpenJUMP.[15]

Unterschied JUMP/OpenJUMP
JUMP wurde schnell ein oft genutztes Open-Source GIS. Eine kleine Reihe von Benutzern begann PlugIns zu entwickeln oder Verbesserungen am Programm selber vorzunehmen. Zwei Entwickler entschieden sich darauf hin das JPP zu bilden, um die eben genannten Erweiterungen und Verbesserungen zu koordinieren. Dadurch sollte die User-Gemeinde profitieren. Der Quellcode von JUMP wurde „geteilt“, und die Version des JUMPs, welche durch das JPP entwickelt wird, als OpenJUMP bezeichnet. Zusätzlich gibt es noch weitere „Versionen“ von JUMP, z.B. DeeJUMP[28].

Weitere Besonderheiten:
OpenJUMP ist unter der LGPL lizenziert. Es unterstützt verschiedene Sprachen, wie z.B. Deutsch, Englisch oder Französisch. Für diese Diplomarbeit wurde die OpenJUMP Version 1.0[29] verwendet.

[27] GUI - Graphical-User-Interface (dt. grafische-Benutzer-Schnittstelle)
[28] Mehr Informationen unter: http://deegree.sourceforge.net/src/demos.html

3.6.2 GNU-LGPL

GeoTools oder auch Deegree[29] stehen unter der GNU Lesser General Public License.

Die GNU *Lesser General Public Licence* (LGPL)[30], zuvor als GNU *Library General Public Licence* bekannt, ist eine etwas entschärfte Variante der GNU *General Public License* (GPL)[31], aber trotzdem in vielen Punkten mit ihr identisch. Steht ein Programm unter der GPL, muss die komplette Software, die dieses Programm nutzt, auch unter der GPL lizenziert werden. Das bedeutet für diese Software, dass sie ebenfalls frei ist.

Die LGPL wurde hauptsächlich für die Entwicklung von Bibliotheken entworfen, d.h. für Sammlungen von Funktionen, die andere Programme nutzen können, daher auch der alte eben erwähnte Name. Es kam zur Namensänderung, weil die Lizenz nicht nur für Bibliotheken gelten sollte. Software, die Funktionen einer LGPL-Bibliothek nutzt, ist gemäß dieser Lizenz kein abgeleitetes Werk und muss demzufolge auch nicht frei sein. Dies bedeutet, dass eine proprietäre Software diese Bibliotheken nutzen könnte. Etwas anderes ist es, wenn die Bibliotheken geändert oder weiterentwickelt werden. Sie müssen dann wiederum unter der LGPL lizenziert werden und sind somit wieder frei.

3.6.3 EPSG

Um Koordinatentransformationen durchführen zu können, werden Transformationsparameter benötigt. Mit einer Koordinatentransformation ist eine Berechnung gemeint, mit welcher die Koordinatenwerte von einem Referenzsystem in ein anderes umgerechnet werden (siehe Koordinatenreferenzsystem nächste Seite). Um den Zusammenhang besser erklären zu können, soll folgendes Szenario helfen. Die Grundlage eines Routenplaners basiert, wie bereits erklärt, auf einem Graphen und einem Routing-Algorithmus. Um eine Route zu ermitteln, werden mindestens Start- und Zielknoten benötigt. Ist aber die gewünschte Start-Position in einem anderen Koordinatenreferenzsystem (z.B. Position durch GPS[32] bestimmt) angegeben als die Knoten des Graphen (Graph wurde z.B. aus Gauß-Krüger-Koordinaten[33] erstellt), muss eine Umrechnung (Transformation) der gegebenen Start-Position in das Koordinatenrefernzsystem des „Graphen“ erfolgen, damit ein

[29] Deegree ist ein Freies Software Projekt, welches Bauteile für den Aufbau von Geodaten-Infrastrukturen zu Verfügung stellt. Mehr Information unter: http://deegree.sourceforge.net

[30] GNU Lesser General Public License 2.1 - Online unter: http://www.gnu.org/licenses/lgpl.html ; Deutsche Übersetzung der LGPL - Online unter: http://www.gnu.de/lgpl-ger.html

[31] Mehr Informationen unter: http://www.fsf.org/licensing/licenses/gpl.html

[32] aus Wkikpedia[34]: GPS - Global Positioning System ist ein satellitengestütztes Navigationssystem des US-Verteidigungsministerium zur weltweiten Positionsbestimmung.

[33] aus Wikipedia[34]: Das Gauß-Krüger-Koordinatensystem ist ein rechtwinkliges Koordinatensystem, das es ermöglicht, jeden Punkt der Erde mit einer Koordinate (Rechts- und Hochwert) eindeutig zu verorten.

möglicher Start-Knoten anhand der Koordinaten gefunden werden kann.

Die *European Petroleum Survey Group* (EPSG) verwaltet einen Katalog von Koordinatenreferenzsystemen und deren Parametern, die für solche Transformationen wichtig sind. Jedes Koordinatenreferenzsystem erhält einen EPSG-Code, wie z.B EPSG:31467 für den 3. Meridianstreifen Gauß-Krüger (DHDN). Diese EPSG-Codes sind von sehr hoher Bedeutung, da sie unter anderem vom OGC verwendet werden. Dieser Katalog wird als *EPSG Geodetic Parameter Dataset* bezeichnet und kann von deren Website heruntergeladen werden. Dabei kann zwischen einer Microsoft Access Datenbank oder zwischen SQL Skript Dateien (mySQL, Oracle oder PostgreSQL) zum Erstellen einer Datenbank eine Auswahl getroffen werden. Seit 2005 ist die EPSG Bestandteil des *Surveying and Positioning Committee der International Association of Oil & Gas Producers* (OGP).[7]

Koordinatenreferenzsystem

Ein Koordinatenreferenzsystem ist ein Koordinatensystem, das durch Verknüpfung mit einem Datum (z.B. einem Geodätischen Datum) auf die reale Welt bezogen ist. Im Falle eines Geodätischen Datums oder Vertikalen Datums handelt es sich bei der realen Welt um die Erde. Der Begriff wurde in der *Norm ISO 19111 Geographic Information - Spatial referencing by coordinates* eingeführt und definiert. Er wird in der Regel mit *CRS* (für *coordinate reference system*) abgekürzt. Ein Synonym ist das mit GML eingeführte *SRS* (für *spatial reference system*).[34]

[34] Eintrag aus GISWIKI - Online unter: http://www.giswiki.org

4 Prototypische Implementierung Route Service

Die Diplomarbeit besteht aus zwei Teilen:

1. Entwicklung der Java-Klassen für Routing und
2. Implementierung der OpenLS Route Service Spezifikation

Abgesehen von der Einarbeitungsphase und der Erstellung einer Testumgebung, um die zu entwickelnden Klassen auf ihre Funktionen hin zu überprüfen, war die Entwicklung der Java-Klassen für das Routing nicht so umfangreich wie die Implementierung der OpenLS Spezifikation.

4.1 Java-Klassen für Routing

Im Kapitel 3 „Grundlagen“ wurden bereits die wichtigen Begriffe für die weitere Beschreibung erklärt und in der Einleitung wurde erwähnt, dass bei der Entwicklung der Klassen für das Routing auf bereits vorhandene Bibliotheken zurückgegriffen werden soll und diese gemäß den Anforderungen anzupassen sind.

GeoTools bietet bereits ein Interface (Schnittstelle), um verschiedene Graph-Typen zu erstellen, mit welchen es auch möglich ist, durch andere angebotene Klassen das Routing (*„Shortest-Path-Finder“*) zu ermöglichen. Im Folgenden sollen nun die div. Änderungen der Klassen und Schnittstellen von GeoTools beschrieben werden.

4.1.1 Graph

Das angebotene Graph Interface von GeoTools wurde in soweit genutzt, dass darauf aufbauend eine eigene Klasse eines Graphen für den späteren Route Service erstellt wurde. Um einen Graphen im Kontext einer Anwendung erstellen zu können, werden zwei Interfaces benötigt, ein GraphBuilder und ein GraphGenerator.

GraphBuilder
Er ist für die Bestandteile des Graphen verantwortlich. Er fügt z.B. neue Objekte (Knoten, Kanten) hinzu oder löscht vorhandene.

GraphGenerator
Er erstellt den eigentlichen Graphen, incl. der Beziehungen der Objekte untereinander.

GraphBuilder und GraphGenerator wurden für die hier benötigten Funktionalitäten erweitert. Sie unterstützen keine Möglichkeiten, Einbahnstraßen in einem Graphen zu erstellen. Wurde also mit dem vorhanden GraphGenerator von GeoTools ein Graph erstellt, wurden keine Einbahnstraßen erstellt, sondern immer Straßen, die in beide Richtungen befahrbar sind. Das Ganze lässt sich noch einmal gut an Abbildung 4.1 zeigen:

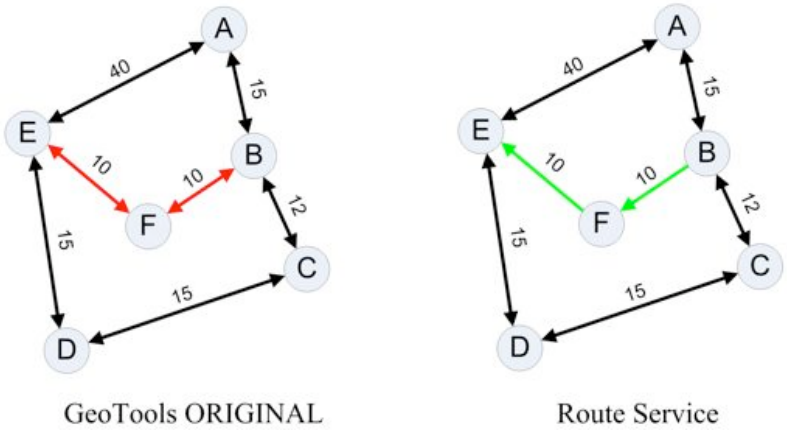


Abbildung 4.1: Graph Original GeoTools - Graph Route Service

Der GraphGenerator und der GraphBuilder wurden so angepasst, dass beim Erstellen einer Kante die Richtung angegeben werden kann. Beim Erstellen einer Kante sind drei Varianten möglich: beide Richtungen, in Richtung und entgegen der Richtung der Geometrie. Letzteres ist wie folgt zu verstehen: hat eine Kante (Linie) ihren ersten Punkt z.B. im Knoten B und ihren letzten in Knoten F, wäre dies die Richtung der Geometrie.

Einbahnstraßen sind bei der Routenplanung für Fahrzeuge sehr wichtig, da diese Straßen im Normalfall (Ausnahme z.B. Feuerwehr) nicht in der verkehrten Richtung befahrbar sind. Für Fußgänger spielt dies allerdings keine Rolle.

4.1.2 Routing-Algorithmus

Der Begriff *Routing*, *Routing-Algorithmus* und der verwendete *Dijkstra-Algorithmus* wurden schon ausführlichst im Kapitel 3 beschrieben. Um gesperrte Straßen beim Routing zu beachten, gibt es verschiedene Arten, dies zu lösen. Eine Möglichkeit wäre, z.B. die

Straße (Kante) einfach aus dem Graphen zu löschen. Somit könnte auf keinen Fall die berechnete Route darüber laufen. Eine andere Variante wäre, die Wichtung der Kante so zu erhöhen, dass der Algorithmus diese Kante beim Routen nicht berücksichtigt. Eine ganz andere Variante, welche allerdings nicht so komplexe Implementierungen erfordert, ist, dem Routing-Algorithmus eine Liste zu geben, in welcher gesperrte Straßen stehen und nicht befahren werden dürfen. Bei dieser „Liste“ handelt es sich genauer gesagt um ein *HashSet()*. Ein HashSet verwaltet Elemente in einer schnellen Hash-basierten Datenstruktur und ermöglicht dadurch ein schnelles finden eines Elementes. Dies ist evtl. nicht die optimalste Lösung dieses Problems, aber auf jeden Fall eine zeitlich umsetzbare.

DijkstraShortestPathFinder

Die Klassen des GeoTools *DijkstarShortestPathFinders* wurden wie folgt angepasst: zum einen, dass sie eine Liste mit gesperrten Straßen entgegen nehmen können, d.h. eine Liste von *Identifikationsnummern* (IDs) der gesperrten Kanten (*EdgeIDs*). Zum anderen musste bei der Ermittlung des *Shortest-Paths* verschiedenes ergänzt werden, damit die gelisteten EdgeIDs nicht in die Berechnung der Route eingehen. Dieses wurde folgendermaßen implementiert: vor jedem Hinzufügen einer Kante zu einer möglichen Route wird geprüft, ob diese Kante (EdgeID) in der Liste der gesperrten Straßen steht. Steht sie in der Liste, wird sie nicht einer möglichen Route hinzugefügt. Das Ganze verdeutlicht die folgende Abbildung noch einmal:

(a) Routing mit Beachtung gesperrter Straßen (b) Routing ohne Beachtung gesperrter Straßen

Abbildung 4.2: Routing mit und ohne gesperrte Straßen

In der Abbildung 4.2 (a) ist sehr gut zu sehen, wie der Algorithmus die Route um die gesperrte Straße (hell-Rot) ermittelt. Die rechte Karte (b) zeigt, wie ein „normaler“ Routenplaner den Weg ermitteln würde, ohne gesperrte Straßen/Gebiete zu berücksichtigen.

Zusammenfassung Graph und Routing-Algorithmus

Durch die erwähnten und noch weiter geänderten und erstellten Java-packages wurden gute Vorraussetzungen geschaffen, um sie für den zu implementierenden Route Service oder auch andere Routenplaner-Anwendungen zu nutzen. Wie schon kurz erwähnt, ist die Umsetzung bzgl. nicht befahrbarer Straßen im Routing-Algorithmus vielleicht nicht die optimalste Lösung. Es wurde aber versucht, mit Hilfe von Schnittstellen zu programmieren, die einen Austausch oder die Erweiterung von Klassen ermöglichen. An dieser Stelle verweise ich auf die Java-Dokumentation, welche sich auf der beiliegenden CD befindet und noch mehr detaillierte Informationen über die Klassen beinhaltet.

Packages:

de.okgis.graph - Implementierung eines eigenen Graphen-Typs - *RSGraph*[1] incl. der wichtigen geänderten GeoTools Schnittstellen des GraphBuilder/GraphGenerators und der daraus resultierenden Klassen RSGraphBuilder und RSGraphGenerator. Siehe Graph Kapitel 4.1.1.

de.okgis.path - ShortestPathFinder von GeoTools in modifizierter Weise, damit eine Liste mit nicht befahrbaren Straßen mitgegeben und beim Routing beachtet werden kann.

de.okgis.graph.traverse.standard - Enthält eine Klasse, in welcher der geänderte Dijkstra's Shortest Path Algorithmus von GeoTools implementiert ist. (lässt beim Routing die nicht befahrbaren Straßen außen vor)

de.okgis.routeservice.routing - Erweiterte Implementierung des ShortestPathFinder. Enthält einige zusätzlichen Funktionen wie z.B.:

- Ermittlung einer ungefähren Reisezeit,
- Längenangabe der kompletten Route,
- Erstellung einer Feature-Collection der kompletten Route, in welcher alle Punkte der Route in Reihenfolge vom Start- zum Zielpunkt sortiert stehen.

[1] RS - Abkürzung für Route Service

4.2 Route Service Architektur

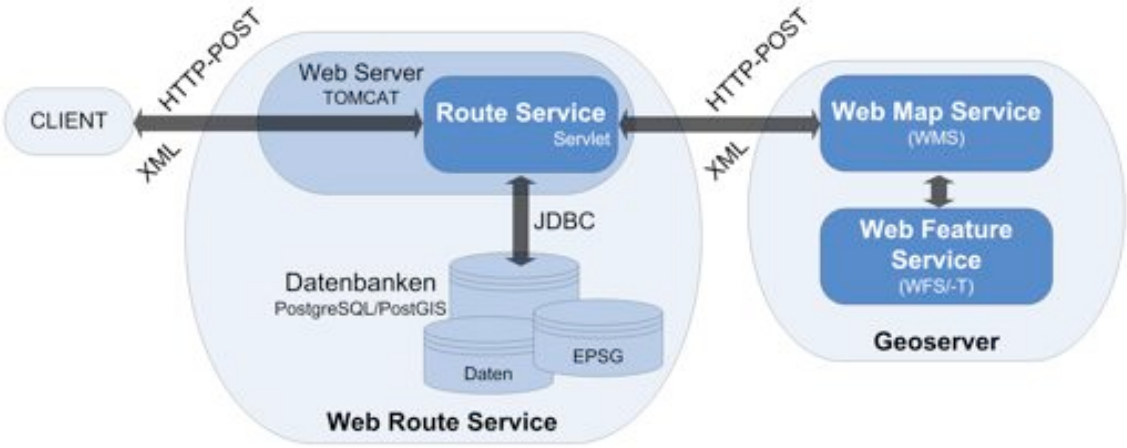


Abbildung 4.3: Prototypische Route Service Architektur

Die Abbildung 4.3 zeigt den Zusammenhang der im Kapitel 3 beschriebenen Grundlagen, um den Route Service prototypisch zu implementieren.

Kurze zusammenfassende Erklärung der Bestandteile:
Um Anfragen an den Route Service senden und ihn natürlich auch testen zu können, wird ein Client benötigt. Anfangs wurde dies durch eine recht einfache Web-Seite, welche über HTTP-POST XML/XLS-Nachrichten versenden kann, umgesetzt.

Der Route Service besteht aus:

- einem Java Servlet, welches auf dem beschriebenen Web Server (Tomcat) läuft und die Anfragen bearbeitet
- verschiedenen Datenbanken (PostgreSQL/PostGIS), um den Graphen zu erstellen, eine Adressbuch- oder POI-Suche durchzuführen und EPSG-Parameter für eine Koordinatentransformation bereitzustellen

Des Weiteren benötigt der Route Service einen WMS, welcher für ihn die Karten mit der eingezeichneten Route erstellt. Dieses wurde für diese Diplomarbeit mit dem Geoserver realisiert und wird ebenfalls über HTTP-POST und XML angesprochen.

4.3 Route Service Realisierung

Im Folgenden soll nun die prototypische Implementierung des Route Services beschrieben werden. Anfangs der Aufbau des Servlets und dessen Initialisierung und anschließend die Realisierung des Routenplaners, welcher sich im Servlet befindet. Die Beschreibung der vielen Optionen des Route Services erfolgt anhand der OpenLS Spezifikation. Im letzten Punkt dieses Kapitels sind zwei Route Requests mit vielen der in diesem Kapitel genannten Parametern und dem dazugehörigen Responses vom Route Service abgebildet.

4.3.1 Servlet

Das Servlet wurde so aufgebaut, dass es im Nachhinein noch ergänzt und somit für weitere Implementierungen von OpenLS Core Services genutzt werden könnte. Es kann in zwei Bestandteile gesplittet werden. Der erste Teil des Servlets übernimmt die Anfrage, die Überprüfung der Anfrage und das Zurückgeben der ermittelten Route. Der zweite Teil des Servlets ist der eigentliche Routenplaner, welcher die Anfrage ausliest, die entsprechende Route ermittelt, eine Antwort erstellt und diese an den ersten Teil des Servlets zurück gibt.

Eine Anfrage an den Route Service erfolgt über XML/XLS-Dokumente. Vor der eigentlichen Bearbeitung dieser Anfragen durch den Routenplaner müssen diese Anfragen überprüft werden. Unter Umständen können sich in solche Dokumente Fehler einschleichen, z.B. falsch geschriebene Parameter oder vergessene Zeichen, wie z.B. "<". Die Überprüfung erfolgt auf zweifache Weise. Zum einen ob das Dokument wohl geformt ist (*well-formed-ness*). Dies bedeutet z.B. ob ein Element jeweils mit dem gleichen Namen endet wie es beginnt oder das keine Zeichen ("<",">") vergessen wurden. Die zweite Prüfung kontrolliert, die Gültigkeit des Dokumentes (*validate*), d.h. es überprüft Struktur und Inhalt der Elemente, ob diese mit dem angegebenen XML-Schema übereinstimmen. Beides wird mit Hilfe von Funktionen, die durch XMLBeans bereitgestellt werden, durchgeführt. Im Falle eines Fehlers in der Nachricht, antwortet der Route Service mit einer Nachricht, in welcher der Fehler und dessen Position steht.

Damit der Route Service funktionsfähig ist, muss dieser konfiguriert und initialisiert werden. Dies geschieht durch eine Funktion (init()) im ersten Teil des Servlets. Der Aufruf erfolgt nach dem Start des Web Servers und vor der Bearbeitung des ersten Requests. Zwei wichtige Dinge sind dort geregelt:

1. Festlegung der Parameter für den Zugriff auf die verschiedenen Datenbanken und
2. Graph für das Routing wird erstellt.

Route Request Sequenzdiagramm

Die Abbildung 4.4 stellt ein Verhaltensdiagramm des Route Services in vereinfachter Form dar. Es zeigt den Weg von einer Route-Anfrage bis zur Rückgabe der ermittelten Route über die verschiedenen Bestandteile des Services. Hiermit kann der erste bzw. zweite Teil des Route Services besser aufgezeigt werden. Der erste Teil des Route Services besteht aus, „RouteService“, RequestOperator und RSListener. Der zweite Teil setzt sich zusammen aus RouteXLSDoc, RouteDocs, Locations, Overlay und Routing.

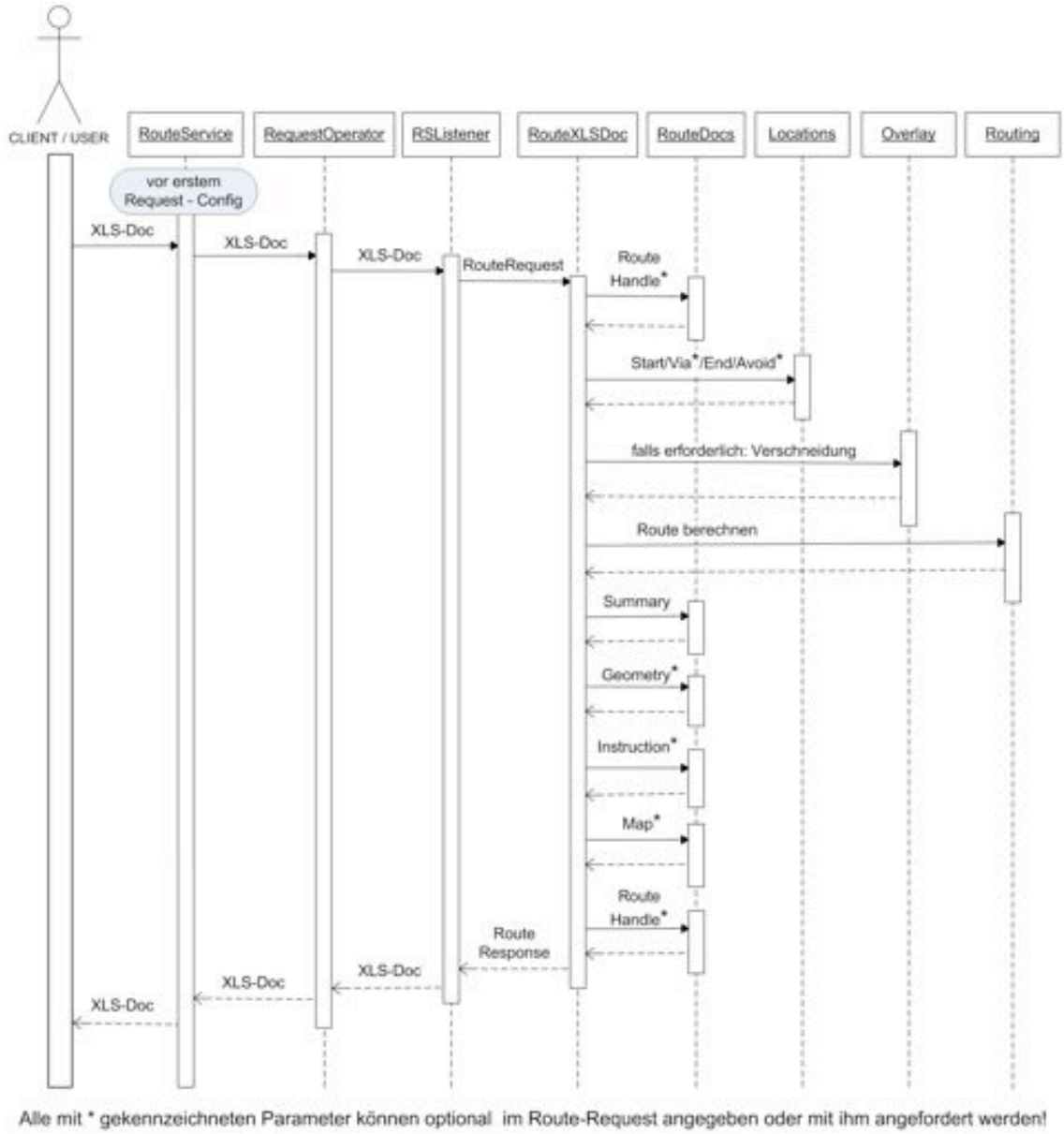


Abbildung 4.4: Sequenzdiagramm des Route Services

4.3.2 XML/XLS-Document

Eine Anfrage und die daraus resultierende Antwort des Route Service erfolgt über ein XML-Dokument, welches XLS enthält. Der Aufbau dieses Dokumentes ist immer ähnlich. Es besteht aus einem Header (Dokument-Kopf) und einem Body (Dokument-Körper).

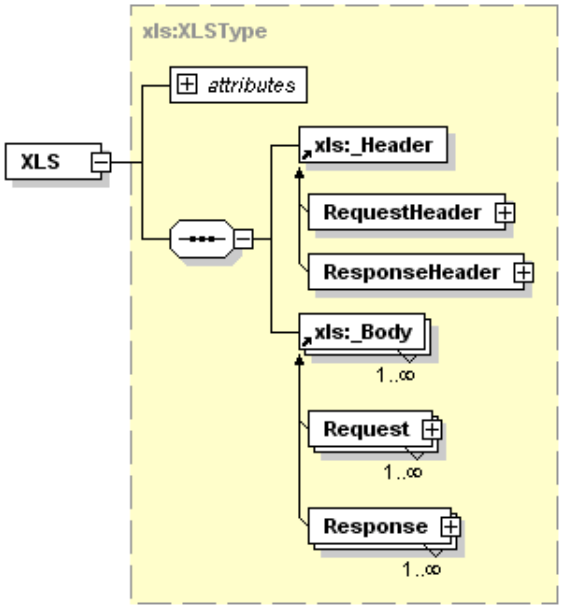


Abbildung 4.5: Schematische Darstellung des XLS-Schemas

In der Abbildung 4.5 ist das XLS-Element als Haupt-Element einer Request oder Response Nachricht zu sehen. Es besteht aus den Attributen *version*, *xls:lang* und den eben erwähnten Header und Body. Diese sind abhängig davon, ob es um eine Anfrage (Request) an oder um eine Antwort (Response) von einem Service geht. Dementsprechend werden sie als RequestHeader und Request und ResponseHeader und Response bezeichnet. Wie in der Darstellung (Abbildung 4.5) zu sehen, kann ein XLS-Element nur einen Header, aber mehrere Bodys enthalten. Somit können mehrere Requests oder Responses in einem XLS-Element erfolgen. Eine Beschreibung der Attribute übernimmt die folgende Tabelle, wobei die letzte Spalte (U=Unterstützt) fortan immer etwas darüber aussagt, ob der implementierte Route Service dieses Attribut unterstützt und nutzen kann.

Tabelle 4.1: XLS-Element - Attribute

Element	Attribut	DatenTyp	E/O	Beschreibung	U
XLS	version	decimal	E	Version des Route Services (*1.1*)	Ja
XLS	xls:lang	language	O	Kürzel für Sprache der Responses (z.B. *de* für Deutsch)	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

RequestHeader

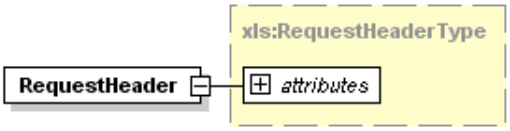


Abbildung 4.6: Schematische Darstellung des RequestHeader-Schemas

Im RequestHeader können verschiedene Attribute angegeben werden. Da diese für die Implementierung des Route Services in seiner ersten Version von nicht sehr hoher Bedeutung sind, wird ein Teil vorerst nicht unterstützt. Im Quellcode wurden alle notwendigen Vorkehrungen dafür getroffen, um sie mit nicht allzu großem Aufwand nachträglich umzusetzen.

Tabelle 4.2: RequestHeader-Element - Attribute

Element	Attribut	DatenTyp	E/O	Beschreibung	U
RequestHeader	clientName	string	O	Name des Clients für Autorisierung	Nein
RequestHeader	clientPassword	string	O	Password von Client	Nein
RequestHeader	sessionID	string	O	Client definierte SessionID, im ResponseHeader angegeben	Ja
RequestHeader	srsName	anyURI	O	Referenz zu einem Koordinatenreferenzsystem	Nein
RequestHeader	MSID	string	O	Ein Client definierte ID, z.B. Gebührenerhebung	Nein
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

Request

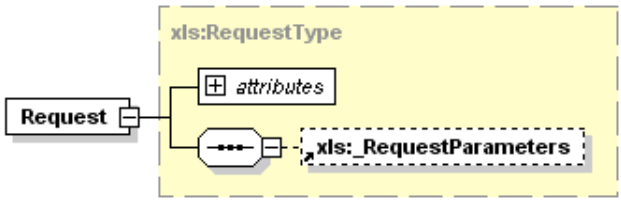


Abbildung 4.7: Schematische Darstellung des Request-Schemas

Vom Request-Element werden alle Attribute, bis auf das *maximumResponses*-Attribut, unterstützt. Es ist für den Route Service nicht wichtig, da dieser nicht mehr Responses geben kann, als Requests erhalten wurden. Dieses Attribut wird erst nützlich, wenn ein Request z.B. an einen Directory Service (Kapitel 2.2.1) geht. Das Request-Element beinhaltet des Weiteren ein Basis Element *xls:_RequestParameters*, welches eine Reihe von

Parametern vertritt. Die Bedeutung dieser verschiedenen Request-Parameter und deren Umsetzung im Route Service wird anschließend beschrieben.

Tabelle 4.3: Request-Element - Attribute

Element	Attribut	DatenTyp	E/O	Beschreibung	U
Request	methodName	NMTOKEN	E	Name der Methode, die vom Service ausgeführt werden soll (*RouteRequest*)	Ja
Request	version	string	E	Version der Request-Parameter vom Clienten (*1.1*)	Ja
Request	requestID	string	E	Client definierte SessionID, steht im Service Response	Ja
Request	maximum-Responses	nonNegativeInterger	O	Erlaubt die Anzahl der Responses zu begrenzen	Nein
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

4.3.3 Request Parameter

DetermineRouteRequest

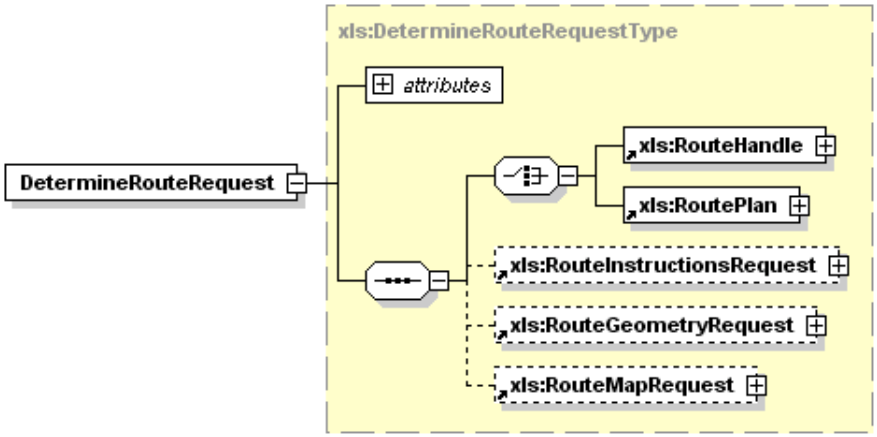


Abbildung 4.8: Schematische Darstellung des DetermineRouteRequest-Schemas

Das DetermineRouteRequest-Element gehört zum Basis Element *xls:_RequestParameters*. Es ist das wichtigste Element und beinhaltet die grundlegenden Parameter in einem Route Request. Bei einem Route Request kann zwischen zwei verschiedenen Arten unterschieden werden: in den *RoutePlan* und den *RouteHandle*. Eine von beiden muss in einem Route Request enthalten sein. Die anderen Parameter, RouteInstructionsRequest, RouteGeoemtryRequest und RouteMapRequest, können optional mit angegeben werden.

Tabelle 4.4: DetermineRouteRequest-Element - Attribute

Element	Attribut	DatenTyp	E/O	Beschreibung	U
*1	provideRouteHandle	boolean	O	Bittet um die Rückgabe eines Route Handles	Ja
*1	distanceUnit	DistanceUnit	O	Kann die Längeneinheit für Response vorgeben (z.B. *KM*)	Ja
*1 DetermineRouteRequest ; E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

Die Bedeutung des *provideRouteHandle*-Attributes wird genauer im Parameter RouteHandle (Kapitel 4.3.3.2) beschrieben.

Mit Hilfe des *distanceUnit*-Attributes kann eine gewünschte Längeneinheit angegeben werden, in welcher z.B. RouteSummary oder die RouteInstruction Ausgabe erfolgen soll. Standard ist „M“ für Meter, weitere Einheiten stehen in der Tabelle.

Tabelle 4.5: DistanceUnit - mögliche Längeneinheiten

Einheit	Beschreibung	Unterstützt
KM	Kilometer	Ja
M	Meter (Standard)	Ja
DM	Dezimeter	Ja
MI	Millimeter	Ja
YD	Yard	Ja
FT	Foot	Ja

4.3.3.1 RoutePlan

Ist das RoutePlan-Element in einem Route Request angegeben, wird mit den darin übergebenen Parametern eine Route ermittelt. Dabei ist es möglich, eine gewünschte Startzeit oder Zielankunftszeit anzugeben.

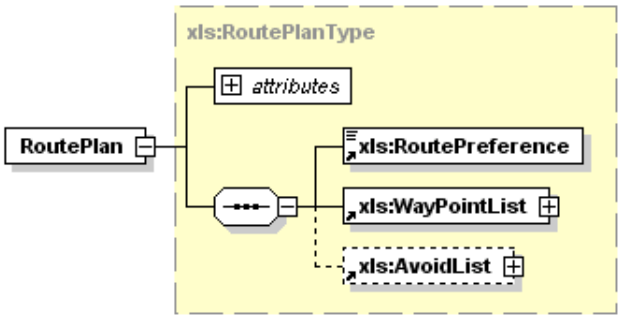


Abbildung 4.9: Schematische Darstellung des RoutePlan-Schemas

Tabelle 4.6: RoutePlan-Element - Attribute

Element	Attribut	DatenTyp	E/O	Beschreibung	U
RoutePlan	useRealTimeTraffic	boolean	O	Beachtung von aktuellen Verkehrsmeldungen bei der Routen-Ermittlung	Nein
RoutePlan	expectedStartTime	dateTime	O	Vorgabe der Startzeit	Ja
RoutePlan	expectedEndTime	dateTime	O	Vorgabe der Ankunftszeit	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

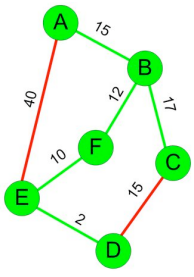
Die weiteren Elemente *RoutePreference*, *WayPointList* und *AvoidList* werden alle vom implementierten Route Service unterstützt und nachfolgend beschrieben:

RoutePreference
Mit diesem Parameter wird die Eigenschaft des Routings angegeben.

Tabelle 4.7: RoutePreference - Möglichkeiten

Art	DatenTyp	Beschreibung	Unterstützt
Fastest	string	schnellste Route (Zeit)	Ja
Shortest	string	kürzeste Route (Strecke)	Ja
Pedestrian	string	zu Fuss	Ja

Im Route Service hat jede Eigenschaft ihre verschiedenen Folgen.
Ist im Route Request *Fastest* angegeben, erhält jeweils die Kante eine Wichtung, die etwas über die Zeit aussagt, welche benötigt wird, um die komplette Kante von Anfang bis Ende zu befahren. Die Zeit wird mit Hilfe einer angegeben Durchschnittsgeschwindigkeit und der Länge der Kante berechnet. Die Länge der Kante und die Anzahl der Kreuzungen werden aber beim Routing nur bedingt berücksichtigt. Es kann also trotzdem passieren, dass nicht unbedingt die schnellste Route ermittelt wurde. Zur weiteren Erklärung noch einmal der Graph aus Kapitel 3.1.3 vom Dijkstra Algorithmus.



Am Graphen links ist gut zu sehen, dass die schnellste Verbindung zwischen A und D, über die Knoten B,F und E verläuft (wenn die Wichtungen der Kanten summiert werden). Allerdings kann dies in Frage gestellt werden. Eine Route über E ist nur unwesentlich länger und bringt zu dem nur einen Knoten mit sich. Um also eine vernünftige Lösung für dieses Problem zu finden, müsste ein Algorithmus implementiert werden, der die Anzahl der Kreuzungen (Ampeln) oder vielleicht auch den Verlauf der Straße zusätzlich beim Routing beachtet. Da dies eine sehr komplexe Sache ist, konnte es nicht im implementierten Route Service umgesetzt werden.

Wird *Shortest* als Route Eigenschaft angegeben, ermitteln sich die Wichtungen der Kanten aus ihrer Länge und das Routing kann mit Hilfe dieser Kriterien problemlos erfolgen.

Für die ersten beiden Arten wird ein gerichteter und gewichteter Graph verwendet. Dagegen wird bei der *Pedestrian* Eigenschaft ein anderer Graph verwendet, denn Fußgänger können auch Einbahnstraßen in entgegengesetzter Richtung laufen. Aus diesem Grund werden bei der Initialisierung des Route Services zwei Graphen erstellt. Einer mit Beachtung und einer ohne Beachtung von Einbahnstraßen.

WayPointList

Durch das WaypointList-Element werden der Start- und Zielpunkt und optional Zwischenpunkte im Route Request dargestellt.

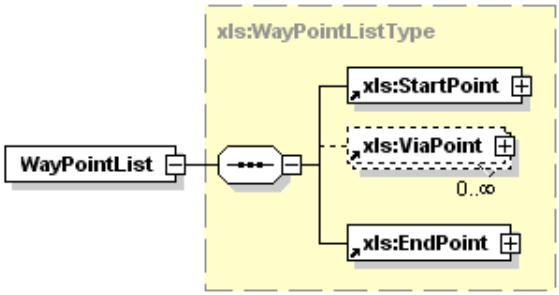


Abbildung 4.10: Schematische Darstellung des WayPointList-Schemas

Wie in der nachfolgenden Tabelle zu sehen, sind diese vom DatenTyp WayPoint.

Tabelle 4.8: WayPointList-Elemente

Element	DatenTyp	E/O	Beschreibung	U
StartPoint	WayPoint	E	Startpunkt der Route	Ja
ViaPoint	WayPoint	O	Zwischenpunkte der Route	Ja
EndPoint	WayPoint	E	Endpunkt der Route (Ziel)	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt				

WayPoint

Das WayPoint-Element enthält neben einem Attribut zwei weitere Elemente. Eines davon ist das Location-Element, dieses wird hier nur kurz erklärt, da es später im Kapitel 4.3.4 ausführlicher beschrieben und deren Umsetzung im Route Service aufgezeigt wird. Es können vier verschiedene Arten von Standorten angegeben werden: eine Adresse, ein Punkt (Koordinaten), eine Fläche oder ein POI.

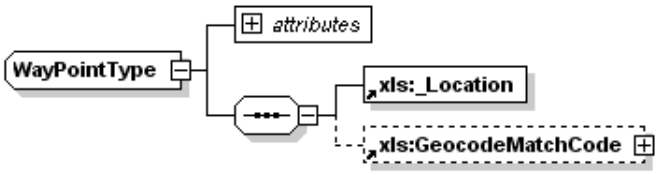


Abbildung 4.11: Schematische Darstellung des WayPoint-Schemas

Der implementierte Route Service unterstützt die Angabe der Zwischenpunkte (ViaPoint). Sie werden also beim Routing mit berücksichtigt, aber in der Reihenfolge angefahren, wie sie angegeben wurden. Weil das Attribut *stop* nicht unterstützt wird, kommt hinzu, dass beim Erreichen der Zwischenpunkte dieses nicht explizit in den Fahranweisungen oder in der Karte angezeigt wird.

Tabelle 4.9: WayPoint-Attribut und dazugehöriges Element

Element	Attribut	DatenTyp	E/O	Beschreibung	U
WayPoint	stop	boolean	O	Möglichkeit am WayPoint zu stoppen	Nein
_Location		AbstractLocation	E	Standorte können sein: Adresse, Punkt (Koordinaten), Fläche oder POI	Ja
GeocodeMatchCode		GeocodingQOS	O	Informationen für Qualität der match-Operationen	Nein
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

AvoidList

Mit Hilfe des AvoidList-Elementes können u.a., die für diese Diplomarbeit wichtigen, zu sperrende Gebiete oder Straßen angegeben werden. Eine ausführlichere Beschreibung und dessen Umsetzung im Route Service erfolgt im Kapitel 4.3.5.

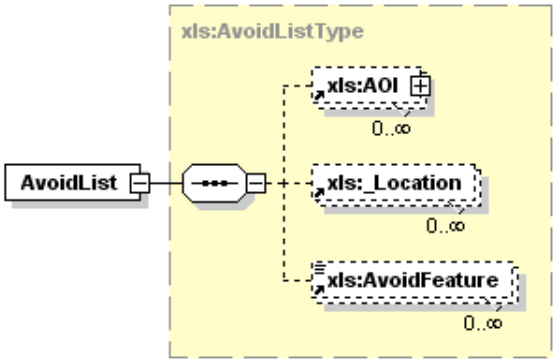


Abbildung 4.12: Schematische Darstellung des AvoidList-Schemas

4.3.3.2 RouteHandle

Zur Erklärung dieses Begriffes muss weiter ausgeholt werden. Bei einem DetermineRouteRequest ist es optional möglich ein Attribut (*provideRouteHanlde*) anzugeben. Wird dieses Attribut bei einem Route Request angegeben und auf *true* gesetzt, wird der Route Request vom Route Service gespeichert. In der Response zum Request wird dann eine RouteID zurückgegeben, mit welcher der gespeicherte Route Request wieder aufgerufen werden kann.

Der Hintergrund des RouteHandle-Elementes ist, laut der OpenLS Spezifikation, dass es möglich sein soll, dieselbe Route, z.B. zu einem anderen Zeitpunkt unter aktuellen Verkehrsmeldungen, wieder neu berechnen zu können, um ggf. eine alternative Route zu erhalten.

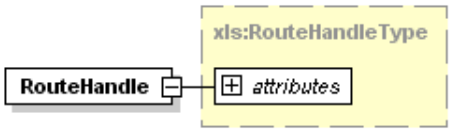


Abbildung 4.13: Schematische Darstellung des RouteHandle-Schemas

Tabelle 4.10: RouteHandle-Element - Attribute

Element	Attribut	DatenTyp	E/O	Beschreibung	U
RouteHandle	serviceID	string	O	ID des Services	Nein
RouteHandle	routeID	string	E	ID der gespeicherten Route	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

Da bei meiner prototypischen Implementierung des Route Services keine Routenplanung unter Nutzung von aktuellen Verkehrsmeldungen möglich ist, war somit etwas Spielraum gegeben. Ich entschied mich für eine etwas andere Umsetzung des RouteHandle-Elementes als es in der OpenLS Spezifikation vorgesehen ist. Dies begründe ich wie folgt. Müssen nach einer Routen-Ermittlung zusätzliche Informationen (Karten, Fahranweisungen etc.) zu einer Route abgerufen werden, kann es vorteilhafter sein, wenn die berechnete Route bereits für den Route Service vorliegt. Ein anderer zusätzlicher Grund für meine Entscheidung war, dass die zu sperrenden Bereiche oder Straßen bei einem Route Request mit angegeben werden müssen und nicht, wie z.B. bei Verkehrsmeldungen, vom Route Service von einem Server oder einer Datenbank abgefragt werden. Bei meiner Implementierung wird also nicht der Route Request (XML-Dokument) sondern die berechnete Route vom Route Service abgespeichert.

Das bedeutet allerdings nicht, dass die Umsetzung, wie sie in der OpenLS Spezifikation steht, nicht zusätzlich programmiert und unter gewissen Änderungen im Route Service einsetzbar ist.

4.3.3.3 RouteInstructionsRequest

Mit Hilfe des RouteInstructionsRequest-Elementes können bei einem Route Request Richtungsanweisungen angefordert werden.

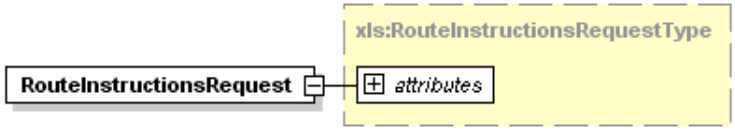


Abbildung 4.14: Schematische Darstellung des RouteInstructionsRequest-Schemas

Die Ausgabe dieser Anweisungen können zusätzlich, durch die Angabe von drei Attributen, erweitert werden.

Tabelle 4.11: RouteInstructionsRequest-Element - Attribute

Element	Attribut	DatenTyp	E/O	Beschreibung	U
*1	format	string	O	Bevorzugtes Format der Fahr- oder Gehanweisung	Ja
*1	provideGeometry	boolean	O	Geometrie zu jeder Fahr- oder Gehanweisung	Ja
*1	provideBoundingBox	boolean	O	Bounding Box zu jeder Fahr- oder Gehanweisung	Ja
*1 RouteInstructionsRequest ; E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

Durch das optionale Attribut *format* kann ein bevorzugtes Format einer Fahr- oder Gehanweisung angegeben werden. Momentan wird aber lediglich *text/plain* unterstützt. Die anderen beiden Attribute, *provideGeometry* und *provideBoundingBox*, sind mit anderen Optionen belegt. Wird das *provideGeometry* Attribut angegeben und auf *true* gesetzt, wird jeder einzelnen Anweisung die Geometrie (Punkt-Koordinaten der Linie(n)) angehängt. Sollte das *provideBoundingBox* Attribut angegeben und auf *true* gesetzt worden sein, wird zu jeder einzelnen Anweisung eine BoundingBox der Linie(n) hinzugegeben.

Die Sprache der Fahranweisungen für Autos und Gehanweisungen für Fussgänger werden über das optionale Attribut *xls:lang* im XLS-Element (Kapitel 4.3.2) angegeben. Standardmäßig erfolgt die Ausgabe in deutsch. Die anderen möglichen Sprachen sind in der nächsten Tabelle zu sehen. Hierbei ist zu erwähnen, dass die Erstellung dieser Anweisun-

gen in anderen Sprachen nur vereinfacht implementiert wurde. Dies bedeutet, es werden jeweils nur die einzelnen Wörter, z.B. *links* mit *left*, getauscht. Das ist nicht die optimalste Lösung, aber aus zeitlichen Gründen wurden die Prioritäten auf andere Teile des Route Services gesetzt.

Tabelle 4.12: xls:lang - optionale Sprachen für Fahr- und Gehanweisungen

Kürzel	Sprache	Unterstützt
de	Deutsch (Standard)	Ja
en	Englisch	Ja
sv	Schwedisch	Ja
it	Italienisch	Ja
es	Spanisch	Ja
fr	Französisch	Ja

Damit die einzelnen „Schritt für Schritt"-Richtungsanweisungen vom Route Service erstellt werden können, wurde ein Algorithmus entwickelt. Dieser Algorithmus nutzt jeweils das letzte Stück der endenden Straße und das erste Stück der nächsten Straße entlang der Route. Mit diesen beiden Linienstücken, die jeweils eine Richtung besitzen, wird der Winkel berechnet unter welchem sie sich in der Verlängerung schneiden. Anhand der Größe und des Vorzeichens dieses Winkels wird anschließend die entsprechende Richtung angegeben. Die folgende Grafik soll das noch einmal verdeutlichen:

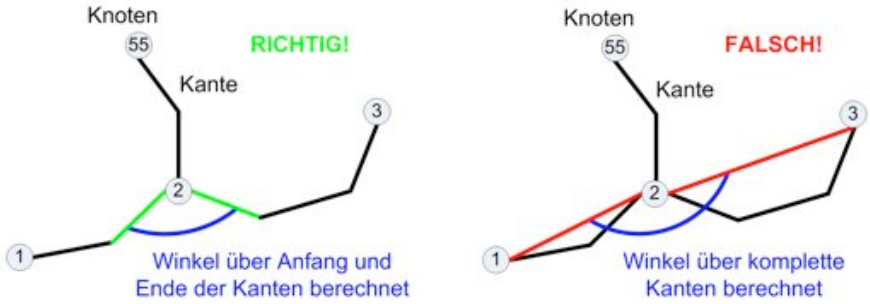


Abbildung 4.15: Intersection-Angle für Richtungsanweisung

Die Abbildung 4.15 macht deutlich, sollte der Schnittwinkel der Kanten 1-2 und 2-3, mit den gesamten Kanten (eine Kante = eine Linie von Anfang bis Ende) berechnet werden, würde eine falsche Richtungsanweisung (z.B. „*Fahren Sie Geradeaus in die XY Straße!*") erfolgen. Die richtige Anweisung ist aber „*Fahren Sie rechts in die XY Straße!*", zu sehen im linken Bild. Der Route Service kann folgende Richtungsanweisungen geben: *rechts*, *halb rechts*, *geradeaus*, *halb links* und *links*. Des Weiteren besteht eine Unterscheidung zwischen Fahr- und Gehanweisung. Die Wahl wird durch die RoutePreference getroffen: „Fahren Sie rechts ..." und „Gehen Sie rechts ...".

4.3.3.4 RouteGeometryRequest

Beim RouteGeometryRequest-Element können zusätzlich und optional drei Attribute angegeben werden. Eins davon ist das *scale* Attribut. Mit ihm kann ein Maßstab angegeben werden, mit welchem ein Wert ermittelt und für die Generalisierung der Route verwendet wird.

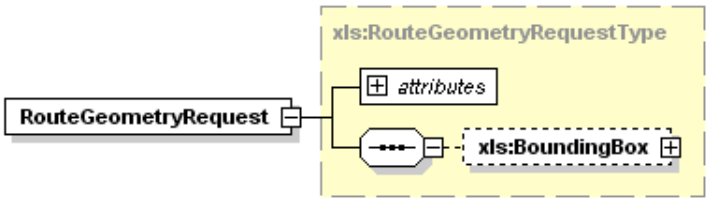


Abbildung 4.16: Schematische Darstellung des RouteGeometryRequest-Schemas

Optional kann die Rückgabe der Geometriepunkte durch die Angabe einer BoundingBox räumlich begrenzt werden. Um durch die angegebene BoundingBox exakte Geometrien zurück geben zu können, muss eine Verschneidung zwischen der Route und der BoundingBox erfolgen. Hierfür wird dieselbe Klasse verwendet, welche für die Verschneidung des Straßennetzes mit den gesperrten Bereichen implementiert wurde. Genauere Informationen werden später noch im Kapitel 4.3.5 beschrieben.

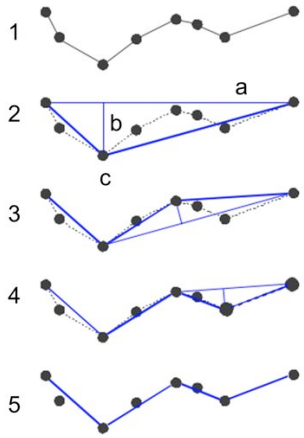
Tabelle 4.13: RouteGeometryRequest-Element - Attribute

Element	Attribut	DatenTyp	E/O	Beschreibung	U
*1	scale	positiveInteger	O	Maximaler Maßstab, mit welchem die Route angezeigt wird, z.B *5000* für „1:5000“	Ja
*1	provideStartingPortion	boolean	O	Wenn *true* angegeben wird, werden ausgehend von der Startposition alle Geometriepunkte innerhalb der BoundingBox, bis zum Erreichen der maxPoints, zurückgegeben	Ja
*1	maxPoints	positiveInteger	O	Maximale Anzahl der zurückgegebenen Punkte (Standard *100*)	Ja
*1 RouteGeometryRequest ; E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

Um die Generalisierung der Route zu ermöglichen, wurde der Douglas-Peucker-Algorithmus nach Bonham-Carter (Geographic information systems for geoscientists. 1994 , GNU FDL) implementiert. Eine genauere Beschreibung dieses Algorithmuses erfolgt auf der nächsten Seite.

Der Douglas-Peucker-Algorithmus ist ein Verfahren zur Kurvenglättung (engl. weeding) bzw. Punktausdünnung. Der Algorithmus findet Anwendung bei der Verarbeitung von Vektorgrafik und der Generalisierung.[2]

Erklärung zum Bild:
- 1: ursprüngliche Linie bestehend aus 8 Punkten
- 2-4: drei Iterationen bis Abstand kleiner als ε
(ε = zuvor festgelegter Grenzwert)
- 5: geglättete Linie bestehend aus nunmehr 5 Punkten
- a = Verbindungslinie zwischen Anfangs- und Endpunkt
- b = maximaler Abstand zwischen Punkt und Linie
- c = Punkt mit maximalem Abstand



Verfahrensschritte:
1. konstruiere Verbindungslinie zwischen Anfangs- und Endpunkt
2. suche Punkt mit maximaler lotrechter Entfernung zu dieser Verbindungslinie
3. wenn Abstand $>$ ε, dann behalte Punkt und gehe zu 1. (nächste Iteration);
wenn Abstand $<$ ε, dann Ende

4.3.3.5 RouteMapRequest und RouteMapOutput

Mit Hilfe der Angabe des RouteMapRequest-Elementes und mindestens einem oder mehreren RouteMapOutput-Elementen können Karten zur berechneten Route angefordert werden. Durch die verschiedenen Attribute des RouteMapOutput-Elementes kann eine Karte den entsprechenden Anforderungen angepasst werden. Wird zusätzlich das optionale Element BBoxContent mit angegeben, kann über eine BoundingBox lediglich ein Ausschnitt der Karte angefordert werden.

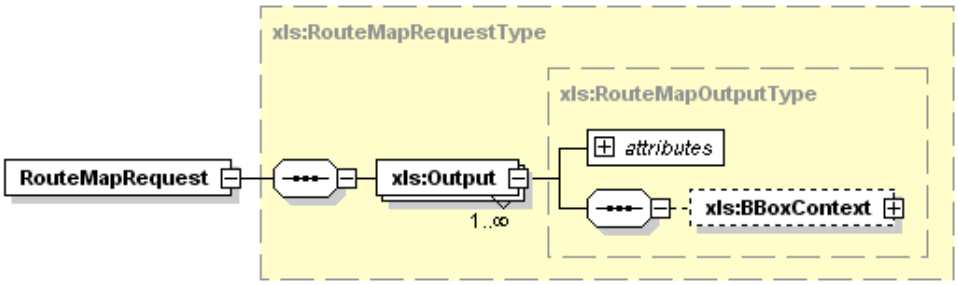


Abbildung 4.17: Schematische Darstellung des RouteMapRequest-Schemas

[2] Eintrag aus Wikipedia[34]

Tabelle 4.14: RouteMapOutput-Element - Attribute

Element	Attribut	DatenTyp	E/O	Beschreibung	U
*1	width	nonNegativeInteger	O	Breite der Karte (Standard *400*)	Ja
*1	height	nonNegativeInteger	O	Höhe der Karte (Standard *400*)	Ja
*1	format	string	O	Ausgabeformat Karte (Standard *png*)	Ja
*1	BGcolor	string	O	Hintergrundfarbe der Karte (Standard *#FFFFFF*)	Ja
*1	transparent	booelan	O	Wenn *true* angegeben, Karte transparent (Standard *false*)	Ja
*1	style	string	O	Style der Karte. Zwei Möglichkeiten: *Overview* oder *Maneuver* (Standard Overview)	Nein
*1 RouteMapOutput ; E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

Die Herstellung der Karten übernimmt der im Kapitel 3.4.3.1 beschriebene WMS des Geoservers. Wie eine normale Karten-Anfrage an einen WMS aussieht und welche Parameter bei einer Anfrage angegeben werden, wurde dort ausführlichst aufgezeigt. Auch die verschiedenen Elemente von SLD wurden im Kapitel 3.4.3.4 erklärt.

Im Route Service wird die Anfrage an den WMS mindestens aus folgenden Parametern zusammengesetzt: NamedLayer (incl. NamedStyle), UserLayer (incl. Userstyles), Breite & Höhe der Karte, Format, BoundingBox und das Koordinatenreferenzsystem der BoundingBox. Speziell die vom Route Service erstellten UserLayer werden kurz beschrieben:

- *UserLayer Route*
  Enthält zum einen die Geometrie (Linie) der Route, sowie deren Darstellung (UserStyle) in der Karte
- *UserLayer START*
  Enthält Koordinaten des Startpunktes, sowie dessen Darstellung und Beschriftung
- *UserLayer END*
  Enthält Koordinaten des Zielpunktes, sowie dessen Beschriftung und einen Link zu einer externen Grafik, die an der Stelle des Zielpunktes dargestellt wird

Wenn AvoidList angegeben, werden weitere UserLayer dem WMS mit übergeben:

- *UserLayer AvoidAreas*
  Layer mit Polygonen der gesperrten Gebiete incl. deren Darstellung
- *UserLayer AvoidAddresse*
  Layer mit Linien der gesperrten Straßen incl. deren Darstellung

Die Anfrage wird über HTTP-POST an den WMS gesendet, dieser erstellt die Karte und sendet sie wieder zurück. Der Route Service nimmt die Karte entgegen und speichert sie in einem zugänglichen Verzeichnis auf einem Server ab.

4.3.4 Location

Das Location-Element beinhaltet drei bzw. vier Möglichkeiten der Angabe eines Standortes. Da diese Möglichkeiten sehr viele Attribute und zusätzliche Elemente beinhalten, wird versucht, dass Ganze möglichst übersichtlich vorzustellen und eine mögliche Unterstützung des Route Services mit anzugeben.

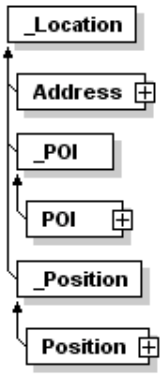


Abbildung 4.18: Schematische Darstellung des Location-Schemas

4.3.4.1 Address

Das Address-Element kann in vielen Elementen, wie z.B. Start- oder Zielpunkt oder in einer AvoidList, angegeben sein. Es besitzt neben zwei Attributen verschiedene Möglichkeiten, eine Adresse anzugeben, eine unstrukturierte freie und eine strukturierte Form.

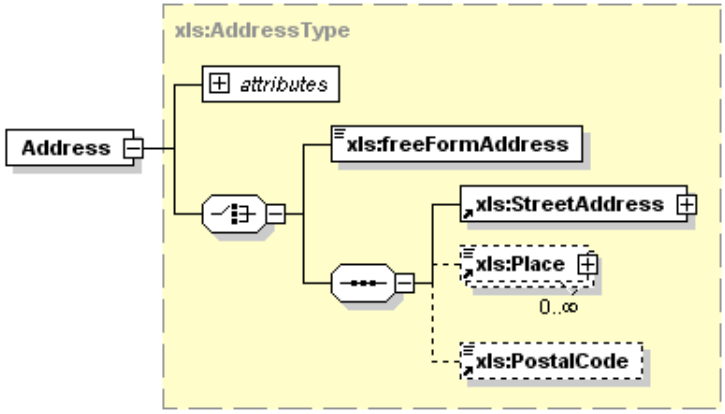


Abbildung 4.19: Schematische Darstellung des Address-Schemas

Tabelle 4.15: Address-Element - Attribute und weitere Elemente

Element	Attribut	DatenTyp	E/O	Beschreibung	U
Address	addressee	string	O	Name des Empfängers	Nein
Address	countryCode	CountryCode	E	Länder-Code nach ISO 3166 Alpha-2	Ja
freeFormAddress		string	E*	Eine unstrukturierte freie Form der Adresse	Ja
StreetAddress		StreetAddress	E*	Eine strukturierte Form der Adresse	Ja
Place		NamedPlace	O	Ein Ort in einer Adresse	Ja
Place	type	NamedPlaceClassification	O	Abgrenzung der verschiedenen Typen eines *Place*	Nein
PostalCode		StreetAddress	O	Postleitzahl	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					
E* = Wahl zwischen zwei Elementen. Eins muss davon angegeben werden.					

Tabelle 4.16: StreetAddress-Element - Attribute und weitere Elemente

Element	Attribut	DatenTyp	E/O	Beschreibung	U
Building	number	string	O	Hausnummer	Ja
Building	subdivision	string	O	Sonstige Unterteilung (z.b. a)	Nein
Building	buildingName	string	O	Gebäudename	Nein
Street	directionalPrefix	string	O	Richtung einer Straße (z.B. Nord)	Nein
Street	typePrefix	string	O	Straßenart vor Straßenname (z.B. Allee)	Nein
Street	officialName	string	O	Straßenname	Ja
Street	typeSuffix	string	O	Straßenart nach Straßenname (z.B. Allee)	Nein
Street	directionalSuffix	string	O	Richtung der Straße nach Straßenname (z.B. Ost)	Nein
Street	muniOctant	CompassPoint	O		Nein
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

Für die Erklärung der Implementierung im Route Service sollen nachfolgend jeweils ein Beispiel für eine unstrukturierte freie und eine strukturierte Form einer Adresse gezeigt werden.

```
<xls:Address countryCode="DE-NI">
        <xls:freeFormAddress>49074 Osnabrück Am Stollenbach 4</xls:freeFormAddress>
</xls:Address>
```

Listing 4.1: freeFormAddress

```
<xls:Address countryCode="DE-NI">
        <xls:StreetAddress>
        <xls:Building number="4"></xls:Building>
                <xls:Street>Am Stollenbach</xls:Street>
        </xls:StreetAddress>
        <xls:Place type="CountrySecondarySubdivision">Osnabrück</xls:Place>
        <xls:PostalCode>49074</xls:PostalCode>
</xls:Address>
```

Listing 4.2: StreetAddress

Die Angabe einer Adresse hat in den Elementen verschiedene Folgen. Ist sie z.B. als Startpunkt angegeben, muss mit Hilfe von ihr eine Position (Koordinaten) ermittelt werden. Das ist die Aufgabe des *Location Utility Service* (Kapitel 2.2.3). Im Route Service wird dies aber intern und nicht über solch einen Service geregelt. Dazu wird in einer Datenbank nach der Adresse gesucht, um die entsprechende Position zu finden.

Die andere Möglichkeit, wo eine Adresse angegeben sein kann, ist in der AvoidList. Dabei soll allerdings keine Position sondern die zur Adresse gehörige EdgeID (ID der Kante) ermittelt werden, damit die entsprechende Straße (Abschnitt) der Adresse gesperrt werden kann.

Eine Besonderheit bei der Implementierung der Suche nach einer Adresse in einer Datenbank ist die Nutzung der Levenshtein-Distanz, welche auch als Edit-Distanz oder Editierabstand bezeichnet werden kann. In der Informationstheorie bezeichnet es ein Maß für den Unterschied zwischen zwei Zeichenketten bezüglich der minimalen Anzahl der Operationen Einfügen, Löschen und Ersetzen, um die eine Zeichenkette in die andere zu überführen. Benannt ist die Distanz nach dem russischen Wissenschaftler Wladimir Levenshtein, der sie 1965 einführte.[34]

Es ist also ein Algorithmus der genutzt werden kann, um z.B. mögliche Rechtschreibfehler bei der Adressangabe zu ignorieren. Sollte demzufolge eine Adresse angegeben und kein passender Eintrag in der Datenbank dazu gefunden werden, kann mit Hilfe dieses Algorithmuses eine möglicherweise gemeinte Adresse ermittelt werden.

Beispiel: Ist *Am Stollenbah* anstatt *Am Stollenbach* angeben, würde zuerst kein Adresseintrag in der Datenbank gefunden werden. Würde anschließend eine genauere Suche mit Hilfe der Levenshtein-Distanz durchgeführt werden, ist die Wahrscheinlichkeit sehr hoch, dass ein entsprechender Eintrag gefunden wird. Die Levenshtein-Distanz zwischen *Am Stollenbah* und *Am Stollenbach* würde 1 betragen, weil ein Buchstabe fehlt. Dieser Algorithmus setzt aber voraus, dass keine groben Fehler in der Adresse enthalten sind. Sollten also mehrere Buchstaben fehlen, wird immer die Adresse verwendet, welche die geringste Distanz zwischen Adressangabe und Eintrag in der Datenbank hat.

Bei Angabe einer *freeFormAddress*, wird durch den Levenshtein-Algorithmus jeweils eine Distanz zwischen der angegebenen Adresse und einem Eintrag in der Datenbank ermittelt. Der Datenbankeintrag (Adresse) mit der geringsten Distanz wird verwendet.

Die Vorgehensweise bei der Angabe einer strukturierten Adresse ist ein wenig anders. Durch die exakte Angabe des Landes, der Postleitzahl, der Stadt, der Straße und der Hausnummer kann eine direkte Abfrage in der Datenbank erfolgen. Sollte kein Eintrag gefunden werden, wird die gleiche Suche wie bei einer *freeFormAddress* ausgeführt.

Die Datenbank (Tabelle) für die Suche einer Adresse musste mit Hilfe des Straßennetzes (Kapitel 3.5 Daten) prototypisch erstellt werden. Dies bedeutet, die Spalte konnte, soweit ein Name vorhanden war, mit den Straßennamen gefüllt werden. Schlechter sah es für die Bereiche der Hausnummern aus. Hierfür lagen leider keine Daten vor und somit konnten nur testweise Hausnummernbereiche in die Adressdatenbank eingetragen werden. Wie genau die Tabelle mit den Adressdaten auszusehen hat, wird im Kapitel 6 „Installationshinweise“ beschrieben.

4.3.4.2 POI

Ein *Point of Interest* (POI) ist ein Punkt, der für einen Nutzer von besonderem Interesse ist. Das könnten z.B. Parkplätze, Kirchen oder Tankstellen sein. Sie können ebenfalls, wie eine Adresse, als Start- oder Zielpunkt und in weiteren Elementen angegeben werden.

Damit der zu implementierende Route Service eine Angabe eines POI unterstützt, musste die Funktionsweise eines *Directory Service* (Kapitel 2.2.1) implementiert werden. Die POI Suche wurde abermals, wie bei der Suche einer Adresse, intern im Route Service implementiert. Aus zeitlichen Gründen und hinsichtlich des Zieles meiner Diplomarbeit, wurde die Umsetzung dieser POI Suche nur vereinfacht realisiert. Nach dem Schema und den Tabellen des POI-Elementes und der weiteren verwendeten Elemente, wird auf die genaue Implementierung im Route Service eingegangen.

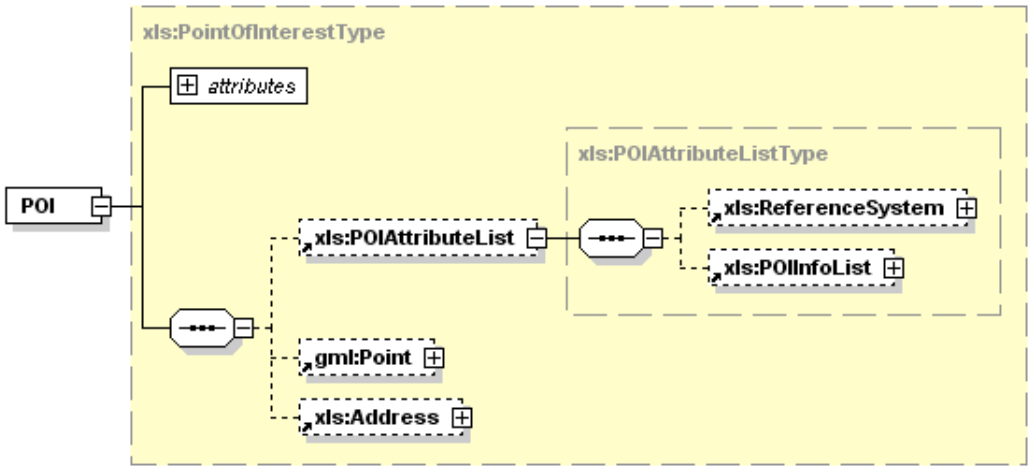


Abbildung 4.20: Schematische Darstellung des POI-Schemas

Tabelle 4.17: POI-Element - Attribute und weitere Elemente

Element	Attribut	DatenTyp	E/O	Beschreibung	U
POI	ID	string	E	ID des POI für Darstellung in Karte	Nein
POI	POIName	string	O	Name des POI für Darstellung in Karte	Nein
POI	phoneNumber	string	O	Telefonnummer des POI für Darstellung in Karte	Nein
POI	description	string	O	Beschreibung des POI für Darstellung in Karte	Nein
POIAttributeList		POIAttributeList	O	Eine oder mehrere formale Namen für Klassifizierungen und/oder POI Informationen	Ja
Point		Point	O	Position (Koordinaten)	Ja
Addresse		Address	O	Address (Siehe Kapitel 4.3.4.1)	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

Tabelle 4.18: POIAttributeList - weitere Elemente (ReferenceSystem)

Element	DatenTyp	E/O	Beschreibung	U
ReferenceSystem	ReferenceSystem	O	Referenzsystem zur Klassifizierung von z.B. Wirtschaftszweigen (Mehr Informationen unter NACE, NAICS oder SIC)	Nein
POIInfoList	POIInfoList	O	Informationen über den POI	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt				

Tabelle 4.19: POIInfoList-Element - weitere Elemente (POIInfo)

Element	Attribut	DatenTyp	E/O	Beschreibung	U
POIInfo	name	string	E	Name des POI	Ja
POIInfo	value	string	E	Eigenschaft des POI (z.B. Fiedhof)	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

Laut der OpenLS Spezifikation gibt es drei Varianten einen POI anzugeben: einmal wird nur das POIAttributeList-Element angegeben, als zweites eine Kombination aus POIAttributeList-Element und Point- oder Address-Element und als dritte Variante nur ein Point- oder Address-Element ohne Angabe eines POIAttributeList-Elementes.

Die Varianten sind allerdings mit dem prototypisch implementierten Route Service nur bedingt nutzbar. Wie in den Tabellen sichtbar, wird nur die Angabe eines POIInfoList-Elementes in einem POIAttributeList-Element unterstützt. Die Kombination aus POIAttributeList- und Point- oder Address-Element ist nicht möglich.

Hier zwei Beispiele, die durch Angabe des POI-Elementes beim implementierten Route Service möglich wären:

```
<xls:POI ID="123456">
  <xls:POIAttributeList>
    <xls:POIInfoList>
      <xls:POIInfo value="Friedhof" name="Friedhof Atter"></xls:POIInfo>
    </xls:POIInfoList>
  </xls:POIAttributeList>
</xls:POI>
```

Listing 4.3: POI POIInfo

```
<xls:POI ID="123456">
  <gml:Point>
    <gml:pos>3433844.00 5796218.15</gml:pos>
  </gml:Point>
</xls:POI>
```

Listing 4.4: POI Point

Wird ein Point- oder Address-Element in einem POI angegeben, wird die angegebene Position im Route Service genutzt. Es findet also keine POI Suche anhand der Koordinaten in einer Datenbank statt.

Sollte ein POIInfo-Element in einem POI angegeben sein, hat dies eine Suche in einer POI Datenbank zur Folge. Hier wird, ähnlich der Suche einer Adresse, mit Hilfe des Namens und der Eigenschaft nach einem möglichen POI gesucht. Ebenfalls, wie bei der Suche

einer Adresse, wird im Falle keines Erfolges die Levenshtein-Distanz verwendet. Dieses schützt allerdings nicht davor, dass evtl. kein Eintrag eines angegebenen POIs vorhanden ist. Hier müsste noch etwas an Arbeit investiert werden, um solch einen Fall abfangen zu können.

Die Grundlagen um eine POI Datenbank (Tabelle) prototypisch zu erstellen, waren, abgesehen von der Vollständigkeit, ganz gut (siehe Kapitel 3.5 Daten). Um mögliche Performanceverluste einsparen zu können, wurden die Daten der POIs im Vorfeld modifiziert. Hintergrund: Ist ein POI als Startpunkt angegeben, wird die Position (Koordinaten) von ihm benötigt, um ihn beim Routing zu verwenden. Diese Koordinaten sagen aber nur etwas über den Standpunkt des POI aus. Es muss also anschließend anhand dieser Koordinaten ein Knoten gesucht werden, welcher beim Routing verwendet werden kann. Um den zuletzt genannten Arbeitsschritt einsparen zu können, wurden bereits in der Tabelle zum jeweiligen POI die nächstmöglichen Koordinaten eines Knoten eingefügt. Zusätzlich wurde die nächstgelegene EdgeID zum POI hinzugefügt. Hierfür wurde eigens eine zusätzliche Anwendung von mir entwickelt, welche im Kapitel 5.4 beschrieben wird.

4.3.4.3 Position

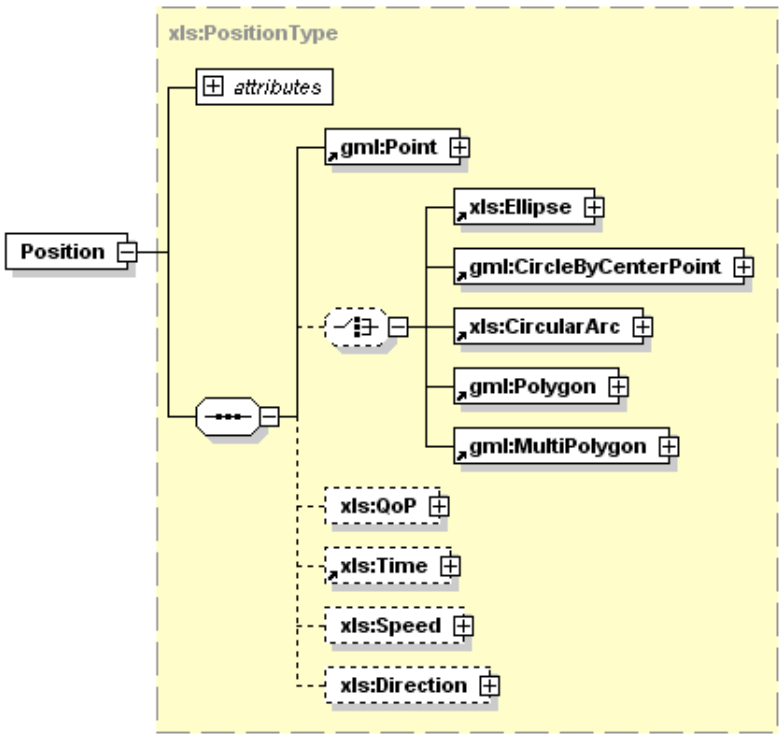


Abbildung 4.21: Schematische Darstellung des Position-Schemas

Durch das Positions-Element kann ein Standpunkt mit Hilfe von verschiedenen Arten von Flächen und durch einen Punkt (Koordinaten) angegeben werden. Das vorangegangene Schema und die jetzt kommende Tabelle zeigen die verschiedenen Möglichkeiten auf.

Tabelle 4.20: Position-Element - Attribute und weitere Elemente

Element	Attribut	DatenTyp	E/O	Beschreibung	U
Position	levelOfConf	string	O	Vertrauensniveau	Nein
Point		Point	E	Punkt (Koordinaten)	Ja
Ellipse		Ellipse	O*	Ellipse	Ja*
CircleByCenterPoint		CircleByCenterPoint	O*	Kreismittelpunkt und Radius	Ja*
CircularArc		CircularArc	O*	Kreisförmiger Bogen	Ja*
Polygon		Polygon	O*	Polygon	Ja*
MultiPolygon		MultiPolygon	O*	MutliPolygon	Ja*
QoP		QualityOfPosition	O	Qualität der Position	Nein
Time		Time	O	Zeit	Nein
Speed		Speed	O	Geschwindigkeit	Nein
Direction		Angle	O	Richtung (Winkel)	Nein
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					
O* = Wahl zwischen fünf Elementen. Optional kann eines davon angegeben werden.					
Ja* = Der implementierte Route Service unterstützt die Angabe, aber nicht im Sinne der OpenLS Spezifikation.					

Teilweise sind verschiedene Positions-Elemente bei der prototypischen Implementierung des Route Services von nicht sehr hoher Bedeutung. Aus diesem Grund wurden sie etwas vereinfachter umgesetzt, als es in der OpenLS Spezifikation vorgesehen ist. Normalerweise soll es optional möglich sein, zu einem Punkt eine dazugehörige Fläche mit anzugeben. Anhand dieser Fläche und des Punktes soll anschließend ein Knoten gefunden werden.

Im implementierten Route Service wurde dies von mir etwas anders umgesetzt. Sollte z.B. ein Polygon zusätzlich angegeben sein, wird der Mittelpunkt dieser Fläche berechnet und versucht, mit ihm einen Knoten zu finden. Die Suche nach einem Knoten ist somit, zumindest im ersten Moment, nicht durch eine angegebene Fläche begrenzt. Hier schon einmal ein Verweis auf das Konfigurationsfile des Route Services, welches im Kapitel 6.3 beschrieben wird. In dieser Datei ist es u.a. möglich, Längenmaße anzugeben, in welchem Bereich Knoten um die Position gesucht werden sollen.

Generell gilt ...

... die beiden optionalen Attribute *gml:id* und *gid* von Positions-Elementen und anderen Elementen werden **nicht** vom implementierten Route Service unterstützt. Das *srsName* Attribut wird dagegen unterstützt, da es von sehr hoher Bedeutung ist. Mit ihm kann das Koordinatenreferenzsystem angegeben werden, in welchem sich die Koordinaten des Elementes befinden. Ein Beispiel hierfür:

Die Angabe des Startpunktes in GK-Koordinaten.

```
<gml:Point>
        <gml:pos srsName="EPSG:31467">3433844.00 5796218.15</gml:pos>
</gml:Point>
```

Listing 4.5: Point GK Koordinaten

Er könnte aber auch WGS-84[3] Koordinaten angegeben werden.

```
<gml:Point>
        <gml:pos srsName="EPSG:4326">8.0292265 52.2954278</gml:pos>
</gml:Point>
```

Listing 4.6: Point WGS84 Koordinaten

Um die Koordinaten in das benötigte Referenzsystem transformieren zu können, werden die unter Kapitel 3.6.3 erwähnten EPSG Parameter benötigt. Die Transformation erfolgt mit Hilfe verschiedener, durch GeoTools und JTS angebotenen, Interfaces und Klassen.

4.3.5 AvoidList

Im Kapitel 4.3.3.1 wurde bereits auf die AvoidList eingegangen. Im Folgenden sollen alle Möglichkeiten aufgezählt und beschrieben werden, welche für die Nutzung zur Sperrung eines Gebietes oder Straße zur Verfügung stehen. Als letzten Punkt in diesem Abschnitt wird auf die Implementierung der Verschneidung des Straßennetzes mit den gesperrten Gebieten eingegangen.

Um dem implementierten Route Service gesperrte Gebiete mitteilen zu können, gibt es durch das *Area of Interest* (AOI) Element drei Möglichkeiten. Die erste Variante wäre, einen Kreis (Kreismittelpunkt mit Radius) zu übergeben. Als zweite Möglichkeit kann ein Polygon des zu sperrenden Gebietes angegeben werden und die letzte Variante wäre die Angabe einer rechteckigen Hülle. In einem AOI-Element können beliebig viele solcher zu sperrenden Gebiete angegeben werden. Was genau mit diesen angegebenen Gebieten

[3] aus Wikipedia[34]: WGS84 - Das World Geodetic System 1984 ist die geodätische Grundlage des GPS-Systems, der Vermessung der Erde und ihrer Objekte mit NAVSTAR-Satelliten.

passiert, wird anschließend bei der Verschneidung beschrieben. Das folgende Schema zeigt die zur Verfügung stehenden Elemente nochmals auf.

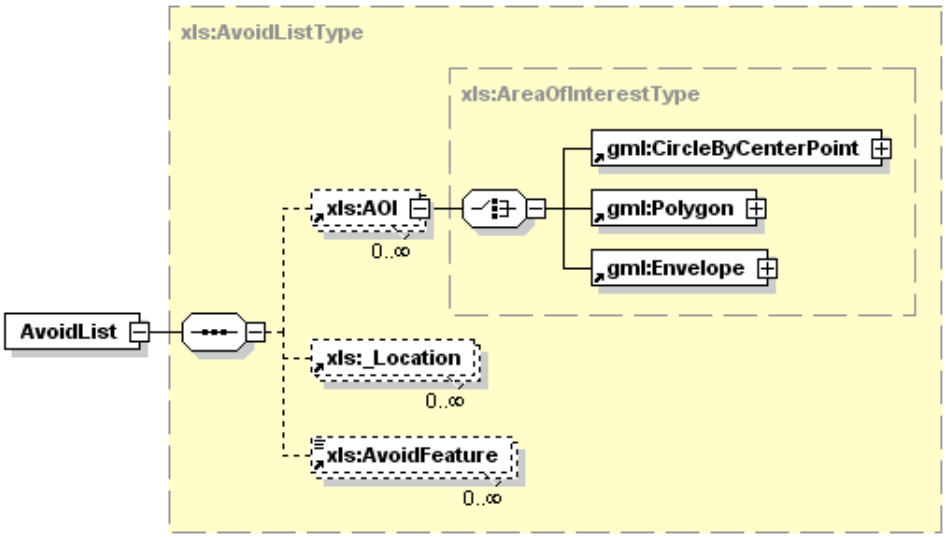


Abbildung 4.22: Schematische Darstellung des AvoidList- und AOI-Schemas

Wie bei einer WayPointList findet auch hier das Location-Element Verwendung.

Tabelle 4.21: AvoidList- und AOI-Element

Element	DatenTyp	E/O	Beschreibung	U
AOI	AreaOfInterest	O	Liste von geographischen Vermeidungsgebieten	Ja
_Location	AbstractLocation	O	Liste von zu vermeidenden Standorten (siehe Kapitel 4.3.4)	Ja
AvoidFeature	AvoidFeature	O	Liste von zu vermeidenden Eigenschaften (z.B Autobahn, Maut)	Ja
CircleByCenterPoint	CircleByCenterPoint	E*	Kreismittelpunkt mit Radius	Ja
Polygon	Polygon	E*	Polygon	Ja
Envelope	Envelope	E*	Rechteckige Hülle, angegeben durch min. und max. Eck-Koordinaten	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt				
E* = Wahl zwischen drei Elementen. Eins muss in einem AOI angegeben sein.				

Die Angabe von Locations wird bei der AvoidList etwas anders gehandhabt als bei der WayPointList (Kapitel 4.3.3.1). War bei der WayPointList noch der Knoten von hoher Bedeutung, ist es bei der AvoidList die Kante (ID/EdgeID der Kante). Ist also z.B. eine Adresse angegeben, wird die dazugehörige EdgeID benötigt, um diese zu sperren. Das

Gleiche gilt für einen POI oder eine Fläche. Die entwickelten Klassen für die Suche einer Adresse oder eines POI können also anstatt einer Position (Koordinaten) auch die ID der Kante, an welcher sie liegen, zurück geben. Sollte eine Position (Koordinaten) zum Vermeiden angegeben sein, muss unterschieden werden, ob diese an einer Kante (Kantenpunkt) oder möglicherweise an einem Knoten liegt. Bei ersterem wird nur die Kante gesperrt. Bei letzterem ist es möglich, einen kompletten Knoten (Kreuzung) mit allen an ihr liegenden Kanten (Straßen) zu sperren. Eine Besonderheit ist, dass bei einer Adressangabe die Hausnummer weggelassen werden kann. Dies hätte zur Folge, dass die komplette Straße incl. aller Kreuzungen gesperrt werden würde.

Eine zusätzliche Option stellt das AvoidFeature-Element bereit. Damit kann, wenn bei einem Route Request gewünscht, die Routenplanung ohne Autobahnen und/oder Mautstraßen erfolgen. Da die Informationen über Autobahnen und Mautstraßen leider nur eingeschränkt zur Verfügung standen, konnten wieder nur testweise Datensätze erstellt werden. Im implementierten Route Service läuft dieser Ausschluß der Autobahnen und/oder Mautstraßen genauso ab, wie die Beachtung der gesperrten Gebiete und Straßen. Wenn eine Routenplanung ohne Autobahnen und/oder Mautstraßen erfolgen soll, werden die entsprechenden EdgeIDs zu der Liste hinzugefügt, welche bei der Routenplanung ausgeschlossen werden.

Verschneidung

Damit die eben erwähnten AOIs genutzt werden können, muss ermittelt werden, welche Kanten durch diese Gebiete betroffen sind. Das erfolgt durch eine Verschneidung des Straßennetzes mit den AOIs (Kapitel 3.1.4 Verschneidung).

Zur Bearbeitung und Erstellung der verschiedenen Daten des Routenplaners wurde OpenJUMP (Kapitel 3.6.1) verwendet. Dieses Programm bietet bereits eine Option zur Verschneidung von Geometrien. Da OpenJUMP ein Open-Source Projekt ist, ist auch der Programmcode verfügbar. Somit konnte die Klasse für die Verschneidung von OpenJUMP für den implementierten Route Service übernommen werden. Innerhalb der Klassen wurden verschiedene Dinge nicht optimal für den Route Service gelöst. In den Originalklassen von OpenJUMP war es z.B. nicht möglich, eine indizierte Feature-Collection zu übergeben. Das wird aber für eine schnelle Verschneidung benötigt. Somit musste dies zusätzlich von mir überarbeitet werden. Die überarbeiteten Klassen werden des Weiteren noch von anderen Teilen des Route Services genutzt (z.B. im RouteGeometryRequest, wo unter Angabe einer BoundingBox nur ein bestimmter Teil der Route zurückgegeben werden kann).

Sind also AOIs in einem Route Request angegeben, werden sie mit dem Straßennetz verschnitten. Hieraus entsteht eine Liste der betroffenen Kanten, die durch ihre EdgeID darin aufgelistet sind. Die Liste wird dann beim Routing von dem im Kapitel 4.1.2 erwähnten *ShortestPathFinder* verwendet, um eine Route unter Berücksichtigung dieser gesperrten Kanten zu ermitteln.

Hier soll schon einmal vorweg genommen werden, dass PostGIS auch eine Funktionalität zur Verschneidung von Geometrien bereitstellt. Dazu aber mehr in der Zusammenfassung und im Ausblick (Kapitel 7).

4.3.6 Response Parameter

Die folgenden Schemata und Abbildungen sollen einen Überblick geben, in wie fern fast alle Response Parameter incl. deren wichtigsten Attribute unterstützt werden.

RouteHandle
Siehe Kapitel 4.3.3.2 RouteHandle.

RouteSummary
Die RouteSummary ist eine Zusammenfassung von Informationen über die Route. Sie wird in jedem Route Response angegeben.

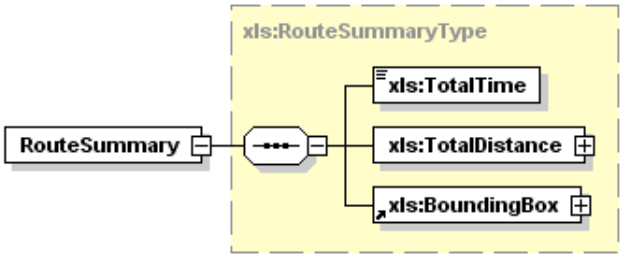


Abbildung 4.23: Schematische Darstellung des RouteSummary-Schemas

Tabelle 4.22: RouteSummary-Element - weitere Elemente

Element	Attribut	DatenTyp	E/O	Beschreibung	U
TotalTime		duration	E	Gesamtzeit	Ja
TotalDistance		Distance	E	Gesamtstrecke	Ja
TotalDistance	value	decimal	E	Gesamtzeit	Ja
TotalDistance	accuracy	decimal	O	Genauigkeit der Gesamtstrecke	Nein
TotalDistance	uom	DistanceUnit	O	Längeneinheit, z.B. „KM“	Ja
BoundingBox		Envelope	E	BoundingBox der Route	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

RouteGeometry
Durch das RouteGeometry-Element wird die Geometrie der Route zurückgegeben.

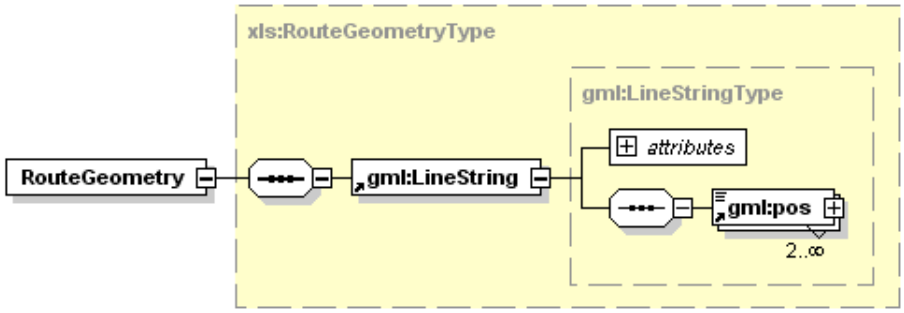


Abbildung 4.24: Schematische Darstellung des RouteGeometry-Schemas

Tabelle 4.23: RouteGeometry-Element

Element	Attribut	DatenTyp	E/O	Beschreibung	U
LineString		LineString	E	Linie mit „allen“ Punkten der Route	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

RouteInstructionsList
Enthält die „Schritt für Schritt“-Anweisungen und weitere Informationen der Route. Das RouteInstructionsList-Element repräsentiert dabei die Gesamtheit aller Anweisungen. Eine einzelne Anweisung erfolgt durch das RouteInstruction-Element.

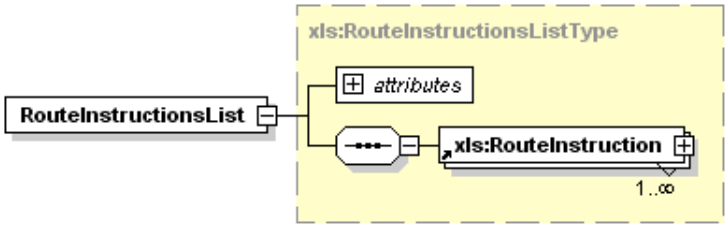


Abbildung 4.25: Schematische Darstellung des RouteInstructionsList-Schemas

Tabelle 4.24: RouteInstructionsList-Element - Attribute

Element	Attribut	DatenTyp	E/O	Beschreibung	U
RouteInstructionsList	format	string	O	Format der A*	Nein
RouteInstructionsList	lang	language	E	Sprache der A*	Ja
RouteInstruction		RouteInstruction	E	Fahr- oder GehA*	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt ; A* = Anweisung					

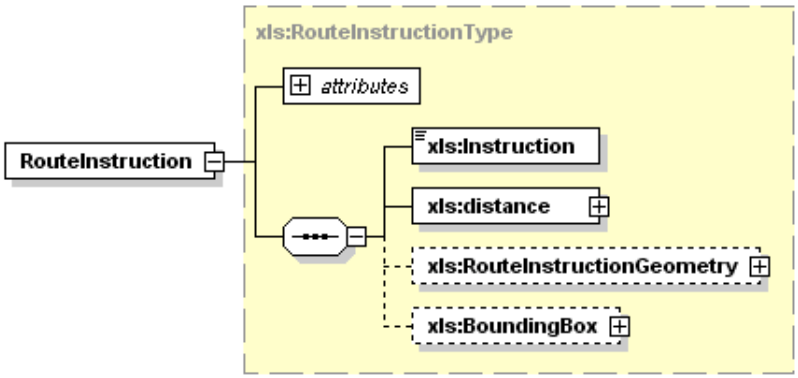


Abbildung 4.26: Schematische Darstellung des RouteInstruction-Schemas

Tabelle 4.25: RouteInstruction-Element - Attribute und weitere Elemente

Element	Attribut	DatenTyp	E/O	Beschreibung	U
RouteInstruction	duration	duration	E	Dauer der A*	Ja
RouteInstruction	description	string	O	Beschreibung	Ja
Instruction		string	E	A*	Ja
distance		distance	E	Distanz der A*	Ja
RouteInstructionGeometry		RouteGeometry	O	siehe RouteGeometry	Ja
BoundingBox		Envelope	O	BoundingBox der A*	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt ; A* = Anweisung					

RouteMap

Durch das RouteMap-Element werden Informationen zur angefragten Karte der Route zurückgegeben.

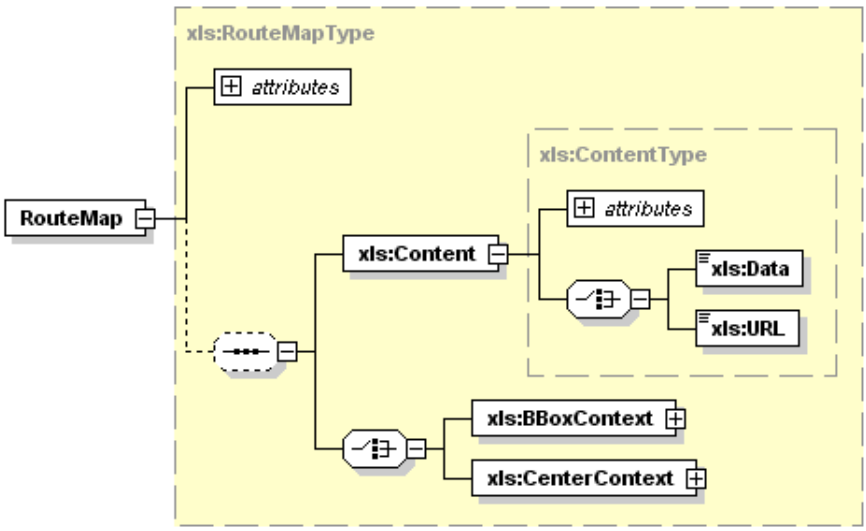


Abbildung 4.27: Schematische Darstellung des RouteMap-Schemas

Tabelle 4.26: RouteMap-Element - Attribute und weitere Elemente

Element	Attribut	DatenTyp	E/O	Beschreibung	U
RouteMap	description	string	O	Beschreibung	Ja
Content		Content	E	Eigenschaften der Route-Map (Karte)	Ja
Content	format	string	E	Format der Karte	Ja
Content	width	integer	E	Breite der Karte	Ja
Content	height	integer	E	Höhe der Karte	Ja
Data		string	E*[1]	Karte in XML	Nein
URL		string	E*[1]	URL zur Karte	Ja
BBoxContext		Envelope	E*[2]	BoundingBox der Karte	Nein
CenterContext		CenterContext	E*[2]	Zentrum Zusammenfassung der Karte	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					
E*[1] = Auswahl zwischen zwei Elementen. Das URL-Element wurde implementiert.					
E*[2] = Auswahl zwischen zwei Elementen. Das CenterContext-Element wurde implementiert.					

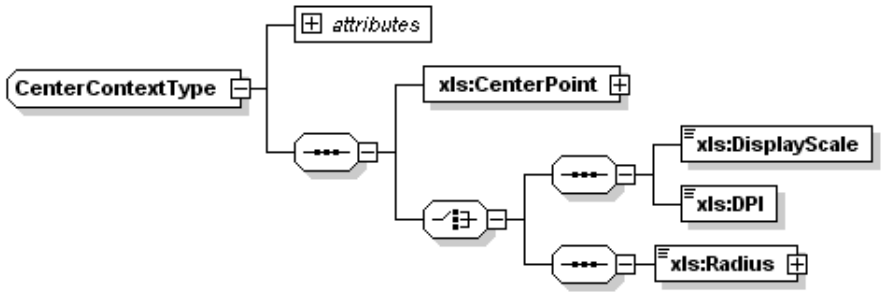


Abbildung 4.28: Schematische Darstellung des CenterContext-Schemas

Tabelle 4.27: CenterContext-Type - Attribute und weitere Elemente

Element	Attribut	DatenTyp	E/O	Beschreibung	U
CenterContext	azimuth	integer	O	Winkel	Ja
CenterContext	SRS	string	E	Koordinatenreferenzsystem (EPSG-Code)	Ja
CenterPoint		Point	E	Mittelpunkt der Karte	Ja
DisplayScale		integer	E*	Anzeigemaßstab des Clients	Nein
DPI		integer	E*	Displayauflösung des Clients	Nein
Radius		Radius	E*	Radius	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					
E* = Auswahl zwischen „zwei“ Elementen (DisplayScale/DPI oder Radius). Das Radius-Element wurde implementiert.					

4.3.7 Request/Response und RouteHandle Beispiel

Im Folgenden soll ein Route Request gezeigt werden, der viele in diesem Kapitel beschriebenen Attribute und Elemente enthält. Danach wird die daraus resultierende Antwort des implementierten Route Services und eine weitere Nutzung durch RouteHandle dargestellt.

Route Request an den Route Service:

```
<xls:XLS xmlns:xls="http://www.opengis.net/xls"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/xls D:\RouteService.xsd"
version="1.1" xls:lang="de">
 <xls:RequestHeader sessionID="987654321"/>
  <xls:Request methodName="RouteRequest" requestID="123456789" version="1.1">
    <xls:DetermineRouteRequest distanceUnit="KM" provideRouteHandle="true">
      <xls:RoutePlan expectedStartTime="2006-08-15T12:12:12">
        <xls:RoutePreference>Fastest</xls:RoutePreference>
        <xls:WayPointList>
          <xls:StartPoint>
            <xls:Position>
              <gml:Point><gml:pos>3434000.00 5796130.00</gml:pos></gml:Point>
            </xls:Position>
          </xls:StartPoint>
          <xls:EndPoint>
            <xls:Position>
              <gml:Point><gml:pos>3433969.00 5796288.00</gml:pos></gml:Point>
            </xls:Position>
          </xls:EndPoint>
        </xls:WayPointList>
      </xls:RoutePlan>
      <xls:RouteInstructionsRequest format="text/plain"
          provideBoundingBox="true" provideGeometry="true"/>
      <xls:RouteGeometryRequest  maxPoints="10"
          provideStartingPortion="false" scale="5000"/>
            <xls:RouteMapRequest>
                    <xls:Output format="png" height="400" width="400"
                                transparent="false" BGcolor="#FFFFFF"/>
            </xls:RouteMapRequest>
          </xls:DetermineRouteRequest>
  </xls:Request>
</xls:XLS>
```

Listing 4.7: Kompletter Route Request

Route Response vom Route Service:

```
<xls:XLS xmlns:xls="http://www.opengis.net/xls"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:gml="http://www.opengis.net/gml" version="1.1"
xsi:schemaLocation="http://www.opengis.net/xls D:\RouteService.xsd">
  <xls:ResponseHeader xsi:type="xls:ResponseHeaderType" sessionID="987654321"/>
  <xls:Response xsi:type="xls:ResponseType" requestID="123456789"
                version="1.1" numberOfResponses="1">
    <xls:DetermineRouteResponse xsi:type="xls:DetermineRouteResponseType">
      <xls:RouteHandle routeID="1154174405531"/>
      <xls:RouteSummary>
        <xls:TotalTime>PT20S</xls:TotalTime>
        <xls:TotalDistance uom="KM" value="0.2"/>
        <xls:BoundingBox>
          <gml:pos>3433917.19 5796136.14</gml:pos>
          <gml:pos>3434011.25 5796288.04</gml:pos>
        </xls:BoundingBox>
      </xls:RouteSummary>
      <xls:RouteGeometry>
        <gml:LineString>
          <gml:pos>3434011.25 5796136.14</gml:pos>
          <gml:pos>3433987.64 5796164.53</gml:pos>
          <gml:pos>3433917.19 5796261.68</gml:pos>
          <gml:pos>3433934.37 5796274.86</gml:pos>
          <gml:pos>3433951.55 5796288.04</gml:pos>
        </gml:LineString>
      </xls:RouteGeometry>
      <xls:RouteInstructionsList xls:lang="de">
        <xls:RouteInstruction duration="PT0S" description="Aktion Nr 1">
          <xls:Instruction>
                Sie starten auf: Mühleneschweg - Tue Aug 15 12:12:12 CEST 2006
          </xls:Instruction>
          <xls:distance value="0"/>
          <xls:RouteInstructionGeometry>
            <gml:LineString>
              <gml:pos>3434011.25 5796136.14</gml:pos>
              <gml:pos>3434011.25 5796136.14</gml:pos>
            </gml:LineString>
          </xls:RouteInstructionGeometry>
          <xls:BoundingBox>
            <gml:pos>3434011.25 5796136.14</gml:pos>
            <gml:pos>3434011.25 5796136.14</gml:pos>
          </xls:BoundingBox>
        </xls:RouteInstruction>
        <xls:RouteInstruction duration="PT10S" description="Aktion Nr 2">
```

```
45          <xls:Instruction>
46                Fahren Sie gerade aus auf: Mühleneschweg für 0.2 KM
47                        - Tue Aug 15 12:12:22 CEST 2006
48          </xls:Instruction>
49          <xls:distance value="0.2"/>
50          <xls:RouteInstructionGeometry>
51            <gml:LineString>
52              <gml:pos>3434011.25 5796136.14</gml:pos>
53              <gml:pos>3433987.64 5796164.53</gml:pos>
54              <gml:pos>3433917.19 5796261.68</gml:pos>
55            </gml:LineString>
56          </xls:RouteInstructionGeometry>
57          <xls:BoundingBox>
58            <gml:pos>3433917.19 5796136.14</gml:pos>
59            <gml:pos>3434011.25 5796261.68</gml:pos>
60          </xls:BoundingBox>
61        </xls:RouteInstruction>
62        <xls:RouteInstruction duration="PT10S" description="Aktion Nr 3">
63          <xls:Instruction>
64                Fahren Sie rechts auf: Römereschstr. für <0.1 KM
65                        - Tue Aug 15 12:12:32 CEST 2006
66          </xls:Instruction>
67          <xls:distance value="0.1"/>
68          <xls:RouteInstructionGeometry>
69            <gml:LineString>
70              <gml:pos>3433917.19 5796261.68</gml:pos>
71              <gml:pos>3433934.37 5796274.86</gml:pos>
72              <gml:pos>3433951.55 5796288.04</gml:pos>
73            </gml:LineString>
74          </xls:RouteInstructionGeometry>
75          <xls:BoundingBox>
76            <gml:pos>3433917.19 5796261.68</gml:pos>
77            <gml:pos>3433951.55 5796288.04</gml:pos>
78          </xls:BoundingBox>
79        </xls:RouteInstruction>
80        <xls:RouteInstruction duration="PT0S" description="Aktion Nr 4">
81          <xls:Instruction>
82                Sie haben Ihr Ziel erreicht! - Tue Aug 15 12:12:42 CEST 2006
83          </xls:Instruction>
84          <xls:distance value="0"/>
85          <xls:RouteInstructionGeometry>
86            <gml:LineString>
87              <gml:pos>3433951.55 5796288.04</gml:pos>
88              <gml:pos>3433951.55 5796288.04</gml:pos>
89            </gml:LineString>
```

```
          </xls:RouteInstructionGeometry>
          <xls:BoundingBox>
            <gml:pos>3433951.55 5796288.04</gml:pos>
            <gml:pos>3433951.55 5796288.04</gml:pos>
          </xls:BoundingBox>
        </xls:RouteInstruction>
      </xls:RouteInstructionsList>
      <xls:RouteMap description="Overview">
        <xls:Content format="png" height="400" width="400">
          <xls:URL>http://localhost:8080/RouteService/routes/1154011333078_0.png
            </xls:URL>
        </xls:Content>
        <xls:CenterContext SRS="EPSG:31467">
          <xls:CenterPoint>
            <gml:pos>3433964.22 5796212.09</gml:pos>
          </xls:CenterPoint>
          <xls:Radius unit="KM">0.08</xls:Radius>
        </xls:CenterContext>
      </xls:RouteMap>
    </xls:DetermineRouteResponse>
  </xls:Response>
</xls:XLS>
```

Listing 4.8: Kompletter Route Response

Mit Hilfe des RouteHandle-Elementes kann die gespeicherte Route wieder abgerufen und zusätzliche Informationen, wie z.B. Karten, RouteInstructions in einer anderen Sprache und Längeneinheit, zu ihr angefordert werden.

Route Request an den Route Service mit RouteHandle:

```
<xls:XLS xmlns:xls="http://www.opengis.net/xls"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/xls D:\RouteService.xsd"
version="1.1" xls:lang="en">
 <xls:RequestHeader sessionID="987654321"/>
  <xls:Request methodName="RouteRequest" requestID="123456789" version="1.1">
    <xls:DetermineRouteRequest distanceUnit="YD">
      <xls:RouteHandle routeID="1154174405531"></xls:RouteHandle>
      <xls:RouteInstructionsRequest format="text/plain"/>
      <xls:RouteMapRequest>
        <xls:Output format="png" BGcolor="#E9C2A6"/>
          <xls:BBoxContext>
            <gml:pos>3433947.19 5796196.14</gml:pos>
            <gml:pos>3433971.25 5796228.04</gml:pos>
```

```
          </xls:BBoxContext>
        </xls:Output>
        <xls:Output format="jpeg" height="500" width="500">
          <xls:BBoxContext>
            <gml:pos>3433017.19 5795236.14</gml:pos>
            <gml:pos>3434911.25 5797188.04</gml:pos>
          </xls:BBoxContext>
        </xls:Output>
      </xls:RouteMapRequest>
    </xls:DetermineRouteRequest>
  </xls:Request>
</xls:XLS>
```

Listing 4.9: Kompletter Route Request mit RouteHandle

Route Response vom Route Service durch Route Handle:

```
<xls:XLS xmlns:xls="http://www.opengis.net/xls"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:gml="http://www.opengis.net/gml" version="1.1"
xsi:schemaLocation="http://www.opengis.net/xls D:\RouteService.xsd">
  <xls:ResponseHeader xsi:type="xls:ResponseHeaderType" sessionID="987654321"/>
  <xls:Response xsi:type="xls:ResponseType"
                requestID="123456789" version="1.1" numberOfResponses="1">
    <xls:DetermineRouteResponse xsi:type="xls:DetermineRouteResponseType">
      <xls:RouteSummary>
        <xls:TotalTime>PT20S</xls:TotalTime>
        <xls:TotalDistance uom="YD" value="219.0"/>
        <xls:BoundingBox>
          <gml:pos>3433917.19 5796136.14</gml:pos>
          <gml:pos>3434011.25 5796288.04</gml:pos>
        </xls:BoundingBox>
      </xls:RouteSummary>
      <xls:RouteInstructionsList xls:lang="en">
        <xls:RouteInstruction duration="PT0S" description="Action nr 1">
          <xls:Instruction>
                You start on: Mühleneschweg - Tue Aug 15 12:12:12 CEST 2006
          </xls:Instruction>
          <xls:distance value="0"/>
        </xls:RouteInstruction>
        <xls:RouteInstruction duration="PT10S" description="Action nr 2">
          <xls:Instruction>
                Drive straightforward on: Mühleneschweg for 171.6 YD
                  - Tue Aug 15 12:12:22 CEST 2006
          </xls:Instruction>
          <xls:distance value="171.6"/>
```

```
30        </xls:RouteInstruction>
31        <xls:RouteInstruction duration="PT10S" description="Action nr 3">
32          <xls:Instruction>
33                Drive right on: Römereschstr. for 47.4 YD
34                  - Tue Aug 15 12:12:32 CEST 2006
35          </xls:Instruction>
36          <xls:distance value="47.4"/>
37        </xls:RouteInstruction>
38        <xls:RouteInstruction duration="PT0S" description="Action nr 4">
39          <xls:Instruction>
40                You arrived at destination - Tue Aug 15 12:12:42 CEST 2006
41          </xls:Instruction>
42          <xls:distance value="0"/>
43        </xls:RouteInstruction>
44      </xls:RouteInstructionsList>
45      <xls:RouteMap description="Overview">
46        <xls:Content format="png" height="400" width="400">
47          <xls:URL>http://localhost:8080/RouteService/routes/1154174663953_0.png
48            </xls:URL>
49        </xls:Content>
50        <xls:CenterContext SRS="EPSG:31467">
51          <xls:CenterPoint>
52            <gml:pos>3433964.22 5796212.09</gml:pos>
53          </xls:CenterPoint>
54          <xls:Radius unit="YD">83.06</xls:Radius>
55        </xls:CenterContext>
56      </xls:RouteMap>
57      <xls:RouteMap description="Maneuver 1">
58        <xls:Content format="jpeg" height="500" width="500">
59          <xls:URL>http://localhost:8080/RouteService/routes/1154174663953_1.jpeg
60            </xls:URL>
61        </xls:Content>
62        <xls:CenterContext SRS="EPSG:31467">
63          <xls:CenterPoint>
64            <gml:pos>3433964.22 5795712.09</gml:pos>
65          </xls:CenterPoint>
66          <xls:Radius unit="YD">1176.66</xls:Radius>
67        </xls:CenterContext>
68      </xls:RouteMap>
69    </xls:DetermineRouteResponse>
70  </xls:Response>
71 </xls:XLS>
```

Listing 4.10: Kompletter Route Response durch RouteHandle

RouteMaps:
Die Abbildung 4.29 ist die RouteMap vom ersten Route Request Beispiel (Listing 4.7 Request und Listing 4.8 Response). Sie zeigt eine Overview-Karte, wie sie standardmäßig vom Route Service zurückgegeben wird.

Abbildung 4.29: Route Request RouteMap

Die anderen zwei Karten (Abbildung 4.30) wurden durch den RouteHandle Request zurückgegeben (Listing 4.9 Request und Listing 4.10 Response). Bei Karte (a) wurde die Hintergrundfarbe geändert und zusätzlich der Kartenausschnitt etwas verkleinert. Die Karte rechts (b) zeigt eine mögliche Übersichtskarte.

(a) RouteMap 1

(b) RouteMap 2

Abbildung 4.30: Route Request mit RouteHandle RouteMaps

Das Anzeigen der Straßennamen und die Darstellung (Farben) erfolgen durch selbst erstellte SLD-Dateien, die auf dem Geoserver abgespeichert sind.

4.3.8 ServiceError

Sollten sich Fehler in einem Route Request befinden oder entstehen Probleme bei der Bearbeitung einer Anfrage, antwortet der implementierte Route Service mit einem ServiceError (ErrorList).

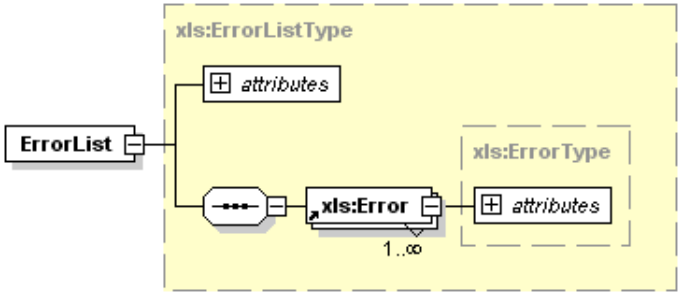


Abbildung 4.31: Schematische Darstellung des ErrorList-Schemas

Tabelle 4.28: ErrorList- und Error-Element - Attribute

Element	Attribut	DatenTyp	E/O	Beschreibung	U
ErrorList	highestSeverity	Severity	O	Größte Schwierigkeit eines Errors	Nein
Error		Error	E	Error	Ja
Error	errorCode	ErrorCode	E	Code des Errors	Ja
Error	severity	Severity	O	Schwierigkeit des Errors	Ja
Error	locationID	IDREF	O	ID des Elementes mit dem Error	Ja
Error	locationPath	string	O	Pfad zum Element oder Attribut des Errors	Ja
Error	message	string	O	Menschenlesbare Erklärung des Errors	Ja
E = Erforderlich(Required) ; O = Optional ; U = Unterstützt					

Folgend ein Beispiel für einen Fehler in einem Route Request.

```
<xls:XLS xmlns:xls="http://www.opengis.net/xls"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.1"
xsi:schemaLocation="http://www.opengis.net/xls D:/XLS.xsd">
  <xls:ResponseHeader xsi:type="xls:ResponseHeaderType" sessionID="Error"/>
  <xls:Response xsi:type="xls:ResponseType" requestID="123456789" version="1.1">
    <xls:ErrorList>
      <xls:Error errorCode="OtherXml" severity="Error"
        message="error: </xls:StartPoin> does not close tag <xls:StartPoint>."/>
    </xls:ErrorList>
  </xls:Response>
</xls:XLS>
```

Listing 4.11: Beispiel für einen ServiceError

5 Zusätzliche Implementierungen

5.1 FileDelete

Weil alle RouteMaps und teilweise auch Routen auf einem Server gespeichert werden, wurde zusätzlich von mir ein kleiner Java Thread[1] entwickelt, der beim Initialisieren des Route Service gestartet wird. Er ist verantwortlich für die Speicherzeit und Löschung der RouteMaps und Routen. Der Thread wird über eine Datei des implementierten Route Services konfiguriert.

Hier ein Auszug aus dem Konfigurationsfile des implementierten Route Services. (Das Konfigurationsfile wird im Kapitel 6.3 Installationshinweise Route Service noch ausführlicher behandelt)

```
1  # *** Parameters for saving and access to RouteMaps and RouteHandle Files ***
2  ROUTEMAPS_PATH=D:/workspace/RouteService/routes
3  ROUTEMAPS_PATH_WWW=http://localhost:8080/RouteService/routes
4  # Parameters for delete Saved Routes
5  # Invoking the thread all .... milsec.
6  # e.g. 1 hour = 3600000 ; 10 min = 600000
7  FILEDELETE_INVOKING=3600000
8  # Delete Files after .... hours.
9  # e.g. 24 hours = 24
10 FILEDELETE_AFTER=24
```

Listing 5.1: Auszug Konfig-File Route Service für FileDelete

In den Zeilen 2-3 wird das Verzeichnis angegeben, in welchen die RouteMaps und Routen gespeichert und von wo sie abgerufen werden können. Zeile 7 und 10 konfigurieren jeweils den Thread, der das Löschen der RouteMaps und Routen übernimmt.
3600000[milsek.] = 1 Stunde
⇒ Thread „schläft“ eine Stunde und überprüft dann, ob Dateien gelöscht werden können.
24 [Stunden]
⇒ 24 Stunden werden RouteMaps und Routen gespeichert

[1] dt. Ausführungsstrang - Prozess der „parallel“ abläuft

5.2 Emergency Route Service

Architektur

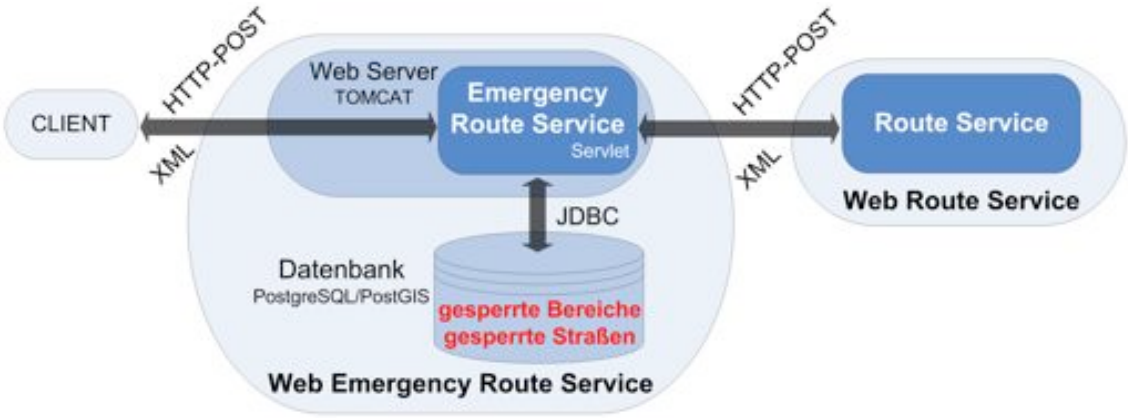


Abbildung 5.1: Emergency Route Service Architektur

Prototypisch wurde zusätzlich noch ein *Emergency Route Service* (ERS) implementiert.

Kurze Erklärung zur Erinnerung
Ein ERS ist ein spezieller Route Service, welcher über ein Netzwerk nutzbar ist und bei der Routenermittlung automatisch aktuelle Gefahrengebiete und gesperrte Straßen berücksichtigt und umfährt.

Wie in der Abbildung 5.1 zu sehen, ist es eine Art Erweiterung des implementierten Route Service. Der ERS wurde mit den gleichen Techniken wie der Route Service erstellt. *Bestandteile:* 1. Java Servlet, 2. PostgreSQL und PostGIS Datenbanken mit aktuellen Gefahrengebieten und gesperrten Straßen (Adressen) und 3. „Route Service OpenLS“.

Realisierung

Anfragen (XML/XLS-Dokument) an den ERS erfolgen genauso wie an den prototypisch implementierten Route Service (OpenLS Route Service). Die Anfragen werden vom ERS entgegengenommen. Aktuelle Gefahrengebiete und gesperrte Straßen werden aus einer DB gelesen und in den Request eingearbeitet. Der komplette Route Request wird anschließend an den implementierten Route Service weitergeleitet. Dieser ermittelt, unter Berücksichtigung der gesperrten Bereiche, die Route und sendet sie zurück.

Mit dieser zusätzlichen Implementierung soll prototypisch die mögliche Funktionsweise eines ERS gezeigt werden. Es wurde eine gute Grundlage geschaffen, um den Service noch weiter zu entwickeln. Ein Teil konnte nur vereinfacht realisiert werden. Alle Da-

tensätze, die in der DB stehen, werden in den Route Request eingearbeitet, auch wenn die mögliche Route überhaupt nicht in deren Reichweite liegt. Möglicher Lösungsweg: Vor der Einarbeitung der AvoidList eine BoundingBox mit Start- und Zielpunkt (ggf. Via-Punkte) erzeugen und mit den gesperrten Bereichen verschneiden. Nur die durch die Verschneidung erhaltenen Gefahrengebiete und Straßen in den Request einarbeiten.

Vereinfachtes Beispiel für einen Route Request, mit automatisch eingefügten Gefahrengebieten und gesperrten Straßen, vom ERS an den Route Service:

```
<xls:XLS ... >
  <xls:RequestHeader/>
  <xls:Request methodName="RouteRequest" requestID="1" version="1.1">
    <xls:DetermineRouteRequest>
      <xls:RoutePlan>
        <xls:RoutePreference>Fastest</xls:RoutePreference>
        <xls:WayPointList> ... </xls:WayPointList>
        <xls:AvoidList>
          <xls:AOI>
            <gml:Polygon>
              <gml:exterior>
                <gml:LinearRing xsi:type="gml:LinearRingType">
                  <gml:pos>3434774.787 5794688.517</gml:pos>
                  <gml:pos>3434932.208 5794547.765</gml:pos>
                  <gml:pos>3434678.483 5794249.592</gml:pos>
                  <gml:pos>3434506.247 5794384.788</gml:pos>
                  <gml:pos>3434774.787 5794688.517</gml:pos>
                </gml:LinearRing>
              </gml:exterior>
            </gml:Polygon>
          </xls:AOI>
          <xls:Address xsi:type="xls:AddressType" countryCode="DE-NI">
            <xls:StreetAddress>
              <xls:Building xsi:type="xls:BuildingLocatorType" number="50"/>
              <xls:Street officialName="Am Stollenbach"/>
            </xls:StreetAddress>
            <xls:Place type="Municipality">Osnabrück</xls:Place>
            <xls:PostalCode>49074</xls:PostalCode>
          </xls:Address>
        </xls:AvoidList>
      </xls:RoutePlan>
    </xls:DetermineRouteRequest>
  </xls:Request>
</xls:XLS>
```

Listing 5.2: Beispiel Route Request von ERS an Route Service (vereinfacht)

5.3 Weboberfläche für Route Service und Emergency Route Service

Damit der implementierte OpenLS Route Service und Emergency Route Service einfacher von einem Nutzer verwendet werden kann, wurde zusätzlich eine Weboberfläche entwickelt. Mit dieser ist es auf einfache Art und Weise möglich, Routen vom Route Service und vom Emergency Route Service planen zu lassen.

Architektur

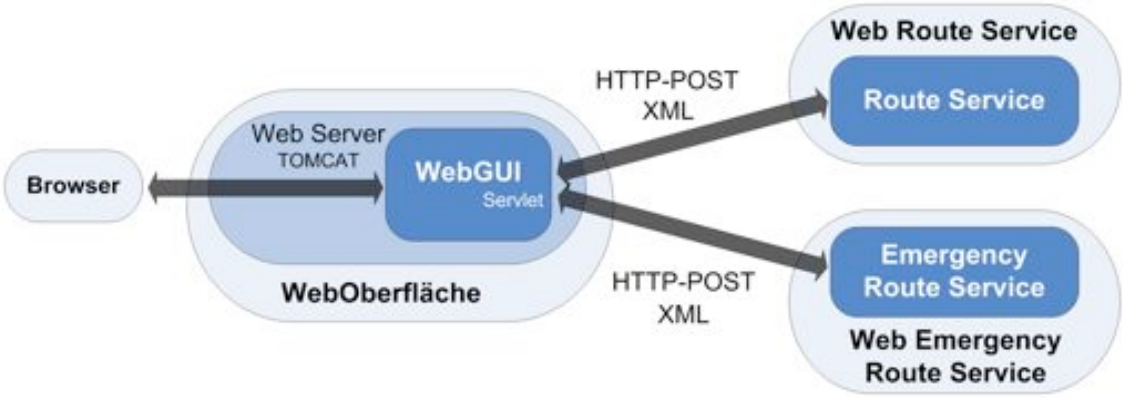


Abbildung 5.2: Weboberfläche Architektur für ERS und RS

Realisierung

Die prototypisch entwickelte Weboberfläche ist ebenfalls ein Java Servlet, welches auf einem Tomcat Server läuft. Das Servlet erstellt eine grafische Benutzeroberfläche (HTML). Der Nutzer kann mit Hilfe von ihr eine Route Anfrage an den Route oder Emergency Route Service stellen. D.h., das Servlet liest die angegebenen Parameter der Website aus, erstellt aus ihnen einen Route Request (gemäß der OpenLS Spezifikation) und sendet den Request an den entsprechenden Service. Die Antwort des Services wird empfangen und formatiert auf der Webseite angezeigt.

Die entwickelte Weboberfläche ermöglicht nur die Nutzung eines Teils der Optionen des OpenLS Route Services. Es können z.B. nur Koordinaten in einem System oder Adressen angegeben werden. Die Nutzung von POIs, Zwischenpunkten oder die Ausgabe der Geometrie der Route ist zur Zeit noch nicht möglich. Die Dokumentation im Quellcode konnte leider aus zeitlichen Gründen nicht so ausführlich, wie bei den anderen Implementierungen, erfolgen.

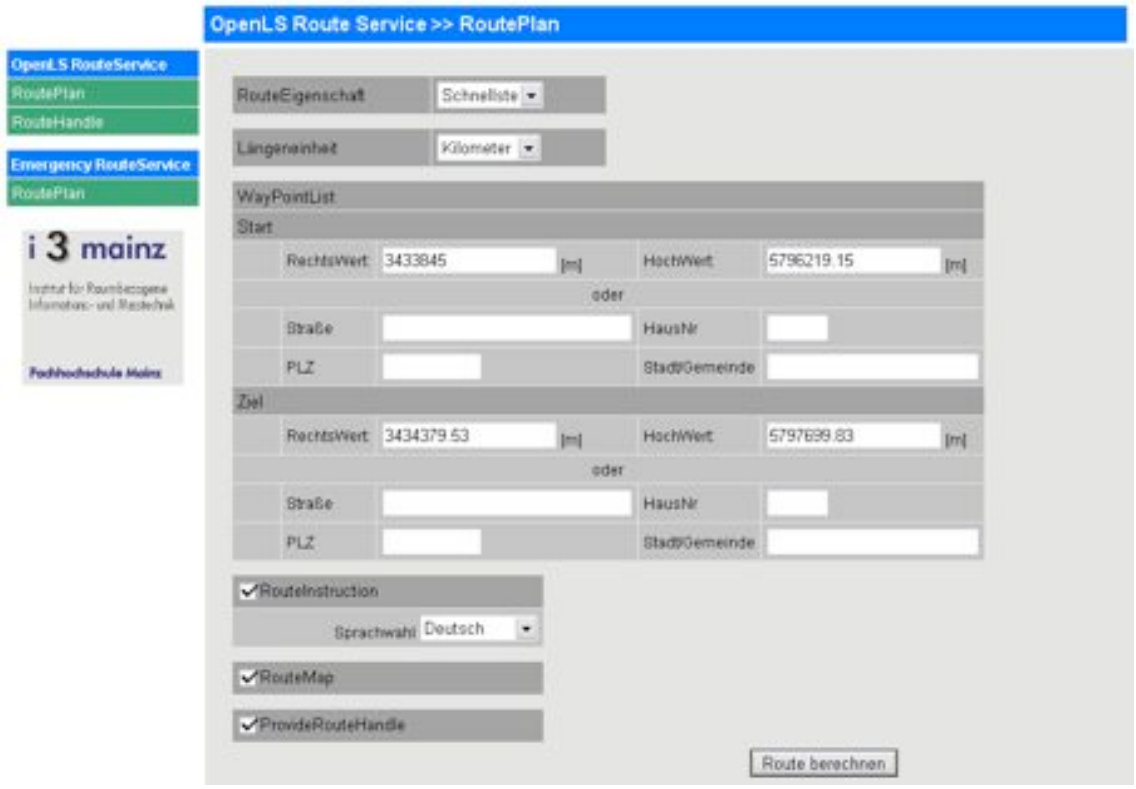


Abbildung 5.3: Weboberfläche - Route Request

In der Abbildung 5.3 ist die RoutePlan-Seite für die Anfrage an einen OpenLS Route Service zu sehen. Das Design und der Inhalt der Route-Plan Seite für die Anfrage an einen Emergency Route Service ist identisch mit dieser.

Abbildung 5.4: Weboberfläche - Route Handle

Die Abbildung 5.4 zeigt die Webseite, um zusätzliche Informationen von einer gespeicherten Route zu erhalten. Mit Hilfe der RouteHandle-ID und den drei Options-Feldern können z.B. noch RouteInstructions in anderen Sprachen angefordert werden.

Abbildung 5.5: Weboberfläche - Route Response

Die Antwort des OpenLS Route oder Emergency Route Services wird formatiert auf der Webseite ausgegeben (Abbildung 5.5).

Abbildung 5.6: Weboberfläche - Route Response ERS

Die Abbildung 5.6 ist ein Beispiel für einen Route Response von einem Emergency Route Service. Der Route Request ist identisch mit dem in Abbildung 5.3. Der einzige Unterschied ist nur, dass der Request nicht an den OpenLS Route Service sondern an den Emergency Route Service gesendet wurde. Wird die RouteMap in Abbildung 5.6 mit der aus Abbildung 5.5 verglichen, sind nochmals die Vorteile eines ERS sichtbar.

5.4 EdgeIDToPoint

Um die vom Route Service genutzten Daten etwas zu optimieren, wurde ein kleines aber sehr nützliches Tool von mir entwickelt. Es liest z.B. ein POI und ein Straßennetz Shapefile[2] ein. Die Anwendung sucht zu jedem POI die am Nächsten gelegene Straße (EdgeID und Koordinaten des Straßenpunktes) und Knoten (Koordinaten des Knoten). Diese Informationen werden dann zusätzlich beim POI (Feature) abgespeichert.

Tabelle 5.1: EdgeIDToPoint - Attribute Vorher

Name:	poiID	poiName	typID	typName	Geometry
Datentyp:	Integer	String	Integer	String	Geometry
Inhalt:	1	St. Matthias Kirche	6	Kirche	POINT(3431724 5799243)

Nach Nutzung des EdgeIDToPoint Tools enthält jedes Feature die alten und zusätzlich noch folgende neue Attribute (Spalten).

Tabelle 5.2: EdgeIDToPoint - Attribute Nachher

Name:	poiID . . . Geometry	nEdgeID	nEdgeNode	nEdgePoint
Datentyp:		Integer	String	String
Inhalt:	1 ... POINT(...)	161	3431682.46 5799186.22	3431706.67 5799185.30
Das Kleingeschriebene „n“ bei den Spaltennamen steht jeweils für „nearest“.				

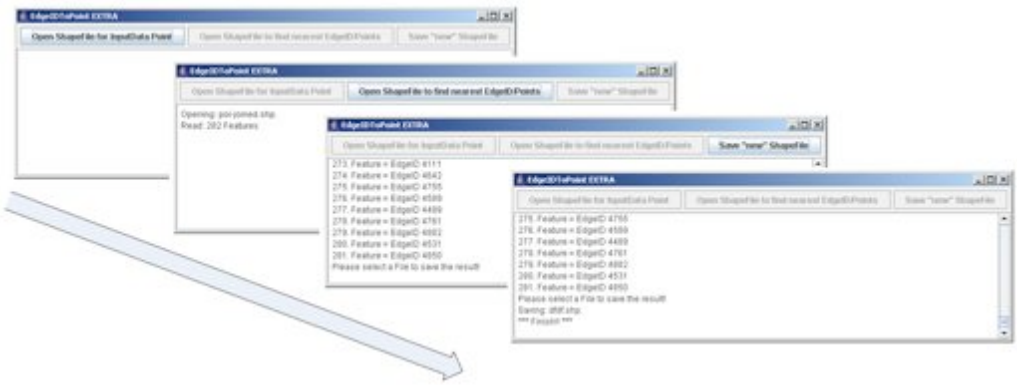


Abbildung 5.7: EdgeIDToPoint - Bedienungsablauf

Das Tool wurde als eine JAR-Datei erstellt und ist durch verschiedene Änderungen, wenn eine aktuelle JRE (Java Runtime Environment) installiert ist, per Doppelklick „ausführbar“. Die Anwendung besteht aus drei Teilen: 1. Daten einlesen 2. Daten ergänzen und 3. Daten speichern. Die Abbildung 5.7 zeigt den Bedienungsablauf der Anwendung.

[2] Ein von „Environmental Systems Research Institute“ (ESRI) entwickeltes Dateiformat für Geodaten. Hat sich mittlerweile zu einer Art „Standard“ für deren Datenaustausch etabliert.

6 Installationshinweise

In diesem Kapitel werden die Vorgänge beschrieben, die für die Installation und Nutzung der prototypischen Implementierungen benötigt werden. Voraussetzung für die Installation des Geoservers und der implementierten Services ist ein installierter Servlet-Container (ab Tomcat 5.5.15 und Java 5.0) und ein PostgreSQL/PostGIS Datenbanksystem.

Im letzten Abschnitt des Kapitels (6.6 Sonstiges) werden die jeweiligen URLs genannt, unter welchen die Services nach ihrer Installation erreichbar sind und angepasst werden können.

6.1 PostGIS Datenbanken

Die prototypisch implementierten OpenLS Route Service und Emergency Route Service benötigten verschiedene Datenbanken. Hierbei handelt es sich immer um PostGIS Datenbanken. Wie PostgreSQL zu installieren ist und PostGIS Datenbanken erstellt werden können, wird hier nicht beschrieben. Die dafür nötigen Anleitungen können in den entsprechenden Webseiten nachgelesen oder von dort heruntergeladen werden.

PostgreSQL - Informationen zum Downloaden, Installieren und Arbeiten mit PostgreSQL online unter: http://www.postgres.de/software.whtml#los

PostGIS - Informationen zum Downloaden und Installieren online unter: http://postgis.refractions.net/download und http://postgis.refractions.net/documentation

Die weitere Beschreibung erfolgt anhand der verwendeten Osnabrücker Frida Daten. Es können aber auch andere Daten verwendet werden.

Route Service

Für den implementierten Route Service ist mindestens eine Datenbank mit zwei Tabellen notwendig. Der Datenbankname sowie die Tabellennamen sind frei wählbar, sie können beliebig im Konfigurationsfile des Route Services (siehe Kapitel 6.3) eingetragen werden.

Für den **Graph** ist eine Tabelle mit mindestens folgenden Spalten notwendig:

Tabelle 6.1: Route Service Graph-Tabelle

Name:	edgeid	strname	oneway	highway	tollway	the_geom
Datentyp:	Integer	String	String	String	String	Geometry
Inhalt:	28	Am Stollenbach	TF	Y	N	MULTILINESTRING(...)

Erklärung der Spalteninhalte von *oneway*, *highway* und *tollway*:

oneway Eigenschaft
„ft“ ⇒ from > to , in Digitalisierungsrichtung
„TF“ ⇒ To > From , gegen die Digitalisierungsrichtung
„y“ oder „ “ ⇒ beide Richtungen erlaubt

highway Eigenschaft	**tollway** Eigenschaft
„Y“ ⇒ ja, Autobahn	„Y“ ⇒ ja, Mautstraße
„N“ ⇒ nein, keine Autobahn	„N“ ⇒ nein, keine Mautstraße
„ “ ⇒ keine Angabe	„ “ ⇒ keine Angabe

Für die Suche einer **Adresse** werden folgende Spalten einer Tabelle benötigt:

Tabelle 6.4: Route Service Address-Tabelle

Name:	edgeid	countrycode	postalcode	place
Datentyp:	Integer	String	Integer	String
Inhalt:	28	DE-NI	49074	Osnabrück
Name:	strname	housenrfrom	housenrto	the_geom
Datentyp:	String	Integer	Integer	Geometry
Inhalt:	Am Stollenbach	100	119	MULTILINESTRING(...)

Beim prototypisch implementierten Route Service wurden die Tabellen Graph und Address zusammengefasst.

Und für die **POI**-Suche ist eine Tabelle mit folgenden Spalten erforderlich:

Tabelle 6.5: Route Service POI-Tabelle

Name:	pooiid	poiname	typid	typname	nedgeid	nedgenode	the_geom
Datentyp:	Integer	String	Integer	String	Integer	String	Geometry
Inhalt:	9	Thomaskirche	6	Kirche	11129	... ...	POINT(...)

Hinweis: Die Spalten *nedgeid* und *nedgenode* können mit dem unter Kapitel 5.4 erwähnten Tool erstellt werden.

Emergency Route Service
Der ERS benötigt ebenfalls eine Datenbank mit zwei Tabellen. Diese müssen wie folgt aufgebaut sein:

Tabelle 6.6: Emergency Route Service AOI-Tabelle

Name:	info	the_geom
Datentyp:	String	Geometry
Inhalt:	Hochwasser	POLYGON(...)

Tabelle 6.7: Emergency Route Service Address-Tabelle

Name:	countrycode	postalcode	place	strname	housenr
Datentyp:	String	Integer	String	String	Integer
Inhalt:	DE-NI	49074	Osnabrück	Insterburger Weg	2

Bemerkung: Ein Wert in der letzten Spalte (*housenr*) ist optional. Enthält sie keine Angabe, wird die komplette Straße incl. Kreuzungen gesperrt sein.

Sollen eigene Daten zum Beispiel aus einem Shapefile verwendet werden, kann die Anpassung an die angegebenen Erfordernisse mit OpenJUMP (Kapitel 3.6.1) erfolgen. Um aus einem Shapefile eine PostGIS Tabelle zu erstellen, gibt es ein Tool („shp2pgsql“), welches sich im PostgreSQL *bin* Ordner befindet.[1]

EPSG
Die EPSG-Daten können unter folgender Webseite heruntergeladen werden:
http://www.epsg.org/Geodetic.html . Mit Hilfe der SQL Skripte werden die erforderlichen Tabellen erstellt. Die Zugangsdaten zu dieser DB müssen im Konfigurationsfile des Route Services eingetragen werden.

6.2 Geoserver

Eine Installationsanleitung für den Geoserver befindet sich unter:
http://docs.codehaus.org/display/GEOSDOC/Quickstart

Verschiedene Dinge müssen dem Geoserver hinzugefügt werden: zum einen die Layer, wie Straßen, Grünflächen, etc., zum anderen die Styles (SLD-Dateien) für die Darstellung der Karte. Am Beispiel der verwendeten Osnabrücker Daten, wurden folgende Layer dem Geoserver hinzugefügt: 1. Straßenlinien, 2. Grünflächen, 3. Gewässerflächen, 4. Gewässerlinien und 5. POIs.

[1] Weitere Informationen unter: http://www.postgis.org/docs/ch04.html#id2523645

Es können jedoch beliebig viele Layer verwendet werden. Dies ist ebenfalls über das Konfigurationsfile des Route Services einstellbar. Zu jedem Layer wurde zusätzlich eine passende SLD-Datei von mir erstellt, welche die Layer je nach Maßstab unterschiedlich darstellt. Diese SLD-Dateien müssen ebenfalls dem Geoserver hinzugefügt werden. Online-Anleitung für das Hinzufügen von:
Layern: http://docs.codehaus.org/display/GEOSDOC/User+Tutorial+Shapefile
SLDs: http://docs.codehaus.org/display/GEOSDOC/SLD+Intro+Tutorial

Die verwendeten Frida Daten (Kapitel 3.5), die von mir erstellten SLD-Dateien und die Anleitungen für den Geoserver (von Geoserver Webseite) befinden sich auf der beiliegenden CD.

6.3 OpenLS Route Service

Die Installation des prototypisch implementierten OpenLS Route Service läuft wie folgt ab: Die sich auf der CD befindende *RouteService.war* muss vor dem Start des Tomcat Servers in dessen *webapps* Verzeichnis kopiert werden. Beim Starten des Tomcat Servers „installiert“ sich anschließend der Route Service von selbst. Wichtig ist dabei, dass vor dem ersten Aufruf des Route Service dieser noch entsprechend den Gegebenheiten konfiguriert wird. Dies passiert über eine Datei (*config.properties*), welche sich im Verzeichnis des Route Services *RouteService/WEB-INF* befindet (Abbildung 6.1 nächste Seite). Sie kann mit einem normalen Text-Editor geöffnet und editiert werden.

Das Konfigurationsfile enthält zu jedem einstellbaren Parameter eine kleine Beschreibung. Auf den nächsten Seiten befindet sich eine zusammengefasste Form der Konfig-Datei (Listing 6.1). Die Parameter für die Verbindungen zu den Datenbanken werden im Listing nur einmal beispielhaft für die Graph-DB dargestellt.

Im Folgenden werden ein paar besondere Parameter aus dem Listing 6.1 beschrieben. Die weiteren Erklärungen können direkt aus dem Listing 6.1 entnommen werden:
Zeile 3&5: Einstellung zum Registrieren von Informationen (wie z.B. wann und von wem erfolgen Requests) oder Fehlern (wie z.B. wurde eine RouteHandleID angefragt, die nicht oder nicht mehr existiert) des Route Service (RS).
Zeile 29 (30): Namen der Layer vom WMS. Sie werden in der Reihenfolge dargestellt wie sie hier angegeben sind. D.h. Grünflächen als unterstes, dann Gewässerflächen usw..
Zeile 48: Die Parameter für das Speichern und Löschen der RouteMaps und Routen wurde bereits im Kapitel 5.1 FileDelete beschrieben.

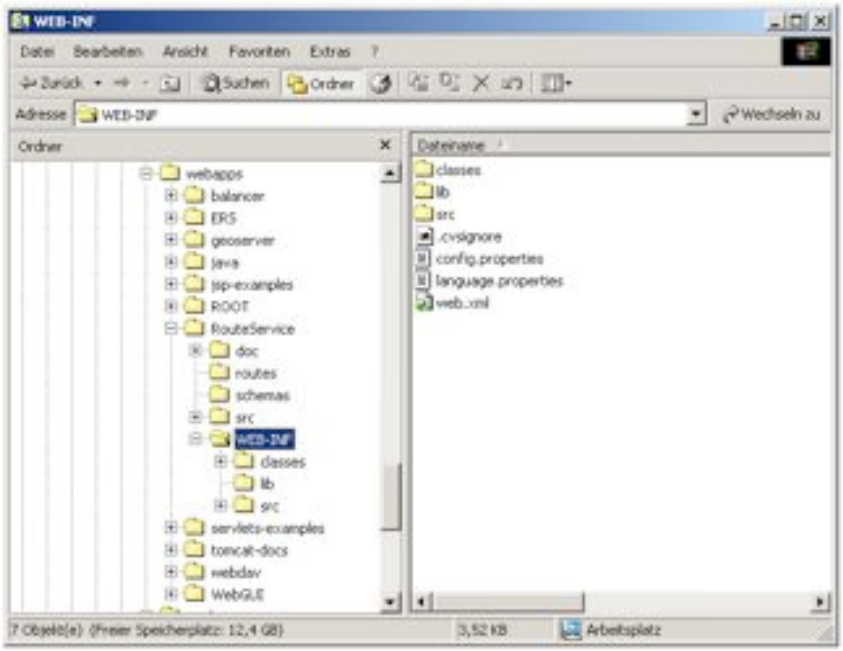


Abbildung 6.1: Verzeichnis von Datei config.properties für Route Service

```
# *** Parameters for Logging ***
# Level for Logging (OPTIONAL)
LOGLEVEL=INFO
# Absolute path to directory where the LogFiles should be located to
LOGDIR=C:/Programme/xampp/tomcat/logs

# *** Parameters for connection to PostGIS Database for Graph ***
PG_HOST=localhost
PG_PORT=5432
PG_DB=RouteService
PG_USER=postgres
PG_PASSWD=postgresAdmin
PG_TABLE=osna
# EPSG-Code for SpatialReferenceSystem (SRS) in which the Graph is
# e.g. "EPSG:31467" for "Gauß-Krüger-Band 3" / GK Band 3
GRAPH_SRS=EPSG:31467

# *** Parameters for connection to PostGIS Database for Address search ***
...
# *** Parameters for connection to PostGIS Database for POI search ***
...
# *** Parameters for connection to PostGIS Database EPSG ***
...

# *** WMS ***
# Parameters for connection to WMS (GeoServer)
```

```
WMS_PATH=http://localhost:8080/geoserver/wms/GetMap
# Name of the Layers
WMS_LAYERS=topp:gruenflaechen-joined,topp:gewaesserflaechen-joined,
                topp:gewaesserlinien-joined,topp:strassen-joined,topp:poi-joined
# Name of the Styles
WMS_STYLES=gruenflaechen,gewaesserflaechen,gewaesserlinien,strassen,pois
# Parameters for Destination-Symbol in RouteMap
# Path
ROUTEMAP_DESTINATION_SYMBOL_PATH=
                http://localhost:8080/geoserver/data/symbols/endflag.gif
# Format
ROUTEMAP_DESTINATION_SYMBOL_FORMAT=image/gif

# *** Language ***
# DefaultResponseLanguage, "de" = German
DEFAULT_RESPONSE_LANGUAGE=de

# *** Parameters for saving and access to RouteMaps and RouteHandle Files ***
ROUTEMAPS_PATH=D:/workspace/RouteService/routes
ROUTEMAPS_PATH_WWW=http://localhost:8080/RouteService/routes
# Parameters for delete Saved Routes
...

# *** Parameters for RouteGeometry ***
# double value wich divides the Scale Parameter in the RouteRequest
#  to get a value for generailze the calculated route
ROUTEGEOM_SCALE=1000

# *** Parameters for RouteSummary & RouteInstruction ***
# double value average speed in meters/second for calculate the TotalTime
# of a route & for calculating the duration for each RouteInstruction
# e.g. 15 for ca. 50km/h
AVERAGESPEED_FASTEST=15
AVERAGESPEED_SHORTEST=15
AVERAGESPEED_PEDESTRIAN=1.5

# *** Parameter to find an Edge/Street ***
# int value of the radius[m] to find an Edge/Street around a given Point/Position
POINT_RADIUS_EDGE=20

# *** Parameter to find a Start/End Position ***
# int value of the maxmimum radius[m] to find a Node around a given Position
POINT_RADIUS_NODE=250
```

Listing 6.1: Konfigurationsfile des Route Service (zusammengefasst)

6.4 Emergency Route Service

Die notwendigen Voraussetzungen für die Installation und die benötigten Datenbanken wurden bereits am Anfang des Kapitels und unter 6.1 beschrieben.

Die Installation des Emergency Route Service erfolgt genauso wie die des implementierten Route Services. Die auf der beiliegenden CD befindende *ERS.war* Datei, muss lediglich in das *webapps* Verzeichnis des Tomcat Servers kopiert werden. Dieser übernimmt die anschließende „Installation“ des Service. Auch der Emergency Route Service besitzt wieder ein Konfigurationsfile, in welchem verschiedene Parameter den Bedürfnissen angepasst werden müssen. Die zu ändernde *config.properties* Datei befindet sich nach dem Start des Tomcat Servers im *tomcat/webapps* Verzeichnis im Ordner *ERS/Web-INF* (ähnlich Abbildung 6.1). Es ist wiederum wichtig, dass das Konfigurationsfile vor dem ersten Request an den Emergency Route Service geändert wurde.

```
# ERS - Emergency Route Service

# Parameters for Logging
# Level for Logging
LOGLEVEL=INFO
# Absolute Path to directory where the LogFiles should be located to
LOGDIR=C:/Programme/xampp/tomcat/logs

# Parameters for connection to PostGIS database :
#  Avoid Polygons and Avoid Addresses
PG_PG=postgis
PG_HOST=localhost
PG_PORT=5432
PG_DB=ERS
PG_USER=postgres
PG_PASSWD=postgresAdmin
PG_TABLE_AOI_POLYGON=aoitoavoid
PG_TABLE_LOCATION_ADDRESS=AdressesToAvoid

# Parameters to connection to RouteService
RS_PATH=http://localhost:8080/RouteService/RouteService

# Parameters SCHEMA_LOCATION_OPENLS (OPTIONAL)
# If not indicated, the provided become used.
# e.g. C:/schemas
SCHEMA_LOCATION=
```

Listing 6.2: Konfigurationsfile des Emergency Route Service

6.5 Weboberfläche für Route Service und Emergency Route Service

Die Weboberfläche lässt sich einfach installieren. Neben den notwendigen Voraussetzungen, die am Anfang des Kapitels beschrieben wurden, muss die in der CD befindende *WebGUI.war* Datei in das *webapps* Verzeichnis des Tomcat Servers kopiert werden. Dieser übernimmt die anschließende „Installation“. Auch bei der WebGUI müssen wieder verschiedene Parameter den Bedürfnissen angepasst werden. Die zu ändernde *config.properties* Datei befindet sich nach dem Start des Tomcat Servers im *tomcat/webapps* Verzeichnis im Ordner *WebGUI/Web-INF* (ähnlich Abbildung 6.1). Es ist erneut wichtig, dass das Konfigurationsfile vor dem ersten Aufruf der Weboberfläche geändert wurde.

```
1 # WebGUI
2
3 # *** ERS-Path ***
4 ERS_PATH=http://localhost:8080/ERS/ERS
5 # *** RS-Path ***
6 RS_PATH=http://localhost:8080/RouteService/RouteService
```

Listing 6.3: Konfigurationsfile der Weboberfläche für RS und ERS

6.6 Sonstiges (URL)

Die Web Services sind im Normalfall unter den folgenden URLs zu erreichen:

OpenLS Route Service (RS) - erreichbar unter:
http://localhost:8080/RouteService/RouteService

Emergency Route Service (ERS) - erreichbar unter:
http://localhost:8080/ERS/ERS

TestClient für RS und ERS - erreichbar unter:
http://localhost:8080/RouteService/testclient-routerequest.html

Weboberfläche für RS und ERS - erreichbar unter:
http://localhost:8080/WebGUI/WebGUI

Alle drei prototypisch implementierten Services können des Weiteren noch zusätzlich angepasst werden. Jeder Service enthält in seinem Verzeichnis einen Ordner *WEB-INF* mit einer Datei Namens *web.xml*. Nachfolgend die *web.xml*-Datei der Weboberfläche:

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
3          "http://java.sun.com/dtd/web-app_2_3.dtd">
4  <web-app>
5          <display-name>Web-GUI for ERS/RS</display-name>
6          <servlet>
7                  <servlet-name>WebGUI</servlet-name>
8                  <servlet-class>de.okgis.webgui.WebGUI</servlet-class>
9          </servlet>
10         <servlet-mapping>
11                 <servlet-name>WebGUI</servlet-name>
12                 <url-pattern>/WebGUI</url-pattern>
13         </servlet-mapping>
14 </web-app>
```

Listing 6.4: Beispiel web.xml Datei der Weboberfläche

Die XML-Datei kann den Bedürfnissen angepasst werden. Änderungen wären z.B. möglich in Zeile 5, 7, 11 und 12.

Zeile 5: Beschreibung des Servlets - wird im Servlet-Container angezeigt

Zeile 7: Name des Servlets (Servlet ID) - wird im Servlet-Container angezeigt

Zeile 11: Servlet ID

Zeile 12: URL Pattern, unter dem das Servlet erreichbar sein soll

Prinzipiell ist eine Web-Anwendung unter:
http://<rechnername>:<Port>/<Web-App Name>/<url-pattern> erreichbar,

d.h. die Weboberfläche ist zu erreichen unter: http://localhost:8080/WebGUI/WebGUI

Würden z.B. folgende Änderungen vorgenommen, in Zeile 12 wird */GUI-OSNA* anstatt */WebGUI* geschrieben, ändert sich die URL wie folgt:
http://localhost:8080/WebGUI/GUI-OSNA

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Das Ziel meiner Diplomarbeit, prototypisch einen OpenLS Route Service zu implementieren, ist in sehr umfangreicher Weise gelungen. Es wurden sogar noch zusätzliche Grundlagen für Erweiterungen geschaffen, wie z.B. eine prototypische Implementierung eines Emergency Route Service oder einer Weboberfläche, um komfortabel Route Requests an einen OpenLS konformen Route Service oder Emergency Route Service zu stellen.

Durch GeoTools war eine Java-Bibliothek vorhanden, aus welcher verschiedene Klassen und Schnittstellen gute Grundlagen boten, um die speziellen Anforderungen des zu implementierenden Route Service zu realisieren.

Der prototypisch implementierte OpenLS Route Service ist eine Web-Anwendung, welche auf einem Tomcat Server läuft. Die Kommunikation findet über HTTP-POST statt und die Anfragen und Antworten erfolgen in XML/XLS-Dokumenten. Die OpenLS Spezifikation stellt eine detaillierte Beschreibung dar, wie der Route Service anzusprechen ist, welche Optionen er bietet und wie die Antworten von ihm aufgebaut sein müssen. Die Spezifikation bringt den Vorteil mit sich, dass neben Start- und Zielpunkt auch eine sogenannte *AvoidList* (dt. Vermeidungsliste) in einem Route Request mit angegeben werden kann. In dieser Liste können Gebiete oder Straßen stehen, durch welche die Route nicht verlaufen soll. Wird in einem Request eine AvoidList angegeben, ermittelt der prototypisch implementierte Route Service die „optimalste“ Route zwischen Start- und Zielpunkt incl. Umfahrung dieser Gebiete.

Bei der Routenplanung können noch weitere Kriterien und Attribute mit angegeben werden. Beispielsweise die Eigenschaft, wie die Route ermittelt werden soll („Schnellste“, „Kürzeste“ oder „Zu Fuß“), oder eine gewünschte Start- oder Zielankunftszeit. Um Informationen über die ermittelte Route zu erhalten, gibt es vier Parameter, welche mit einem Route Request angefordert werden können und anschließend im Route Response stehen.

1. RouteSummary - sind grundsätzliche Informationen, wie Gesamtstrecke und Gesamtzeit der Route.

2. RouteGeometry - sind geometrische Informationen zur ermittelten Route (Linie mit „allen“ Wegpunkten der Route)

3. RouteInstruction - sind „Schritt für Schritt“ Fahr- oder Gehanweisungen der ermittelten Route. Dies wurde in einfacher Form in unterschiedlichen Sprachen implementiert.

4. RouteMaps - sind Karten, in denen die ermittelte Route dargestellt wird. Es können mehrere RouteMaps gleichzeitig angefragt werden, z.B. eine Übersichtskarte oder detaillierte Start- und Zielpunktsansicht der Route.

Bei der Implementierung des Route Service wurde versucht so zu programmieren, dass verschiedene Teile, wie Graph oder Routing-Algorithmus, erweiterbar oder austauschbar sind. Der Hintergrund dafür wird im Kapitel 7.3 „Ausblick“ erläutert. Auf eine gute Dokumentation im und über den Quellcode wurde Wert gelegt, um die Wiederverwendbarkeit und die Erweiterung der erstellten Java-packages zu erhöhen.

Zur Performance des implementierten Route Services:
Leider lagen nur „kleine“ Daten zum Testen der Funktionalität des implementierten Route Services vor. Es konnten somit „nur“ Routenplanungen innerhalb der Stadt Osnabrück getestet werden. Diese absolvierte der Route Service aber in der Regel binnen kürzester Zeit ($<$2 Sekunden, Route Request incl. allen möglichen Optionen & AvoidList). Von großer Bedeutung ist die Rechenleistung und der Arbeitsspeicher des Servers. Interessant wäre es, den Route Service mit größeren Datensätzen, wie z.B. Deutschland und auf einem Server, der nicht *Lokal* läuft, zu testen.

Die Tests aller Implementierungen erfolgten und verliefen fast problemlos auf zwei unterschiedlichen Systemen. PC mit Windows XP Professional mit Service Pack 2 und Laptop mit Windows 2000 Professional und Service Pack 4. Bei den verschiedenen Tests gab es nur zwei kleine Probleme, welche im Folgenden beschrieben werden.

7.2 Offene Probleme

Für Koordinatentransformationen wurden verschiedene Klassen von GeoTools verwendet. Bei der implementierten Transformation, unter Nutzung dieser Klassen, wird von einer dieser Klassen eine *NullPointerException* ausgelöst. Das Verwunderliche ist, dass die Meldung nur beim ersten Nutzen der Klassen (der ersten Transformation nach dem Starten des Route Service) erscheint und dass trotzdem die Transformation durchgeführt wird. Es ergeben sich somit keine Probleme oder Nachteile für das weitere Bearbeiten

dieses oder der folgenden Route Requests. Werden die Klassen für eine Transformation in einer „normalen“ Java-Anwendung verwendet, funktioniert alles ohne jegliche Fehlermeldungen. Dieses Phänomen könnte aber auch wie folgt beseitigt werden: PostGIS bietet u.a. eine Funktion (*transform()*) zum transformieren von Koordinaten an. Es könnte somit eine eigene Klasse implementiert werden, welche mit Hilfe von PostGIS Koordinaten transformiert.

Ein anderes Problem, was aber nichts mit den Implementierungen von mir zu tun hat, wird vom Geoserver verursacht. Dieser löst beim Herunterfahren des Tomcat Servers ebenfalls eine Fehlermeldung aus (*IllegalAccessException*). Recherchen im WWW und Eintragungen in Mailing-Lists brachten aber keine Ergebnisse zur Beseitigung dieser Meldung. Vielleicht lag dies am verwendeten Java 5.0 oder an der Geoserver Vision. Allerdings brachte auch ein Wechsel zur vorherigen Version keine Veränderung.

7.3 Ausblick

Wie bereits in der Zusammenfassung (Kapitel 7.1) erwähnt, wurden verschiedene Teile des Route Service so implementiert, dass sie erweiterbar oder austauschbar sind. Der Hintergrund dafür ist, dass der Graph oder Routing-Algorithmus bei der Verwendung des implementierten Route Service für z.B. ganz Europa oder Deutschland Probleme bereiten könnte. Momentan werden die Wichtungen der Kanten vor jedem Routing berechnet. Besser wäre es z.B., wenn die Wichtungen jeder Kante bei der Initialisierung des Route Service berechnet und in einer Datenbank abgespeichert werden würden. Dazu könnte die Implementierung des Routing-Algorithmus so geändert werden, dass dieser sich lediglich die Wichtungen der Kanten aus der Datenbank holt und die Route berechnet. Würden die Wichtungen der Kanten in einer DB stehen, könnte zu dem die Berücksichtigung der gesperrten Straßen und Gebiete anders erfolgen. In diesem Falle müssten lediglich die Wichtungen der gesperrten Kanten in der Datenbank so angepasst (erhöht) werden, dass die Verwendung der Kanten beim Routing „nicht“ erfolgt. Durch die hohe Wichtung wird die Kante für die Route uninteressant. PostGIS bietet zusätzlich eine Methode zum Verschneiden von Geometrien, dadurch könnte auch die Verschneidung über die PostGIS DB erfolgen.

Der implementierte Route Service unterstützt große Teile der OpenLS Route Service Spezifikation. Das Attribut „useRealTimeTraffic“ konnte leider nicht unterstützt werden. Die Implementierung hätte ohne weiteres erfolgen können, ist aber daran gescheitert, weil es keinen „einfachen“ Zugriff auf die Daten der Verkehrsmeldungen gibt. Recherchen bei Radiosendern und der Polizei (Präsidium für Technik, Logistik und Verwaltung - Lan-

desmeldestelle) brachten keinen Erfolg, um Zugang zu der Datenbank zu erhalten. Die einzigste Möglichkeit wäre gewesen, die Verkehrsmeldungen sozusagen über die „Luft-Schnittstelle“ zu empfangen und selbst in einer Datenbank zu verwalten. Weil dies mit Kosten für einen TMC[1]-Decoder[2] und hohem Aufwand verbunden gewesen wäre, konnte es in meiner Diplomarbeit nicht umgesetzt werden. Hier könnte der Route Service also noch erweitert werden, damit auch Echtzeit-Verkehrsmeldungen bei der Routenplanung verwendet werden können.

Als einen weiteren wichtigen Aspekt sehe ich, dass die Fahr- und Gehanweisungen hinsichtlich ihrer Sprache weiter optimiert werden müssten. Momentan erfolgt ein einfacher Austausch der Worte. Die Fahranweisungen sollten aber entsprechend ihrer Sprache grammatikalisch richtig aufgebaut werden.

Zusätzlich könnten die Anweisungen um Landmarken erweitert werden. Landmarken sind im Zusammenhang mit Routenplanung auffällige oder markante Objekte entlang einer Straße. Dadurch könnte eine bessere Assoziation zwischen einer Anweisung und der Realität hergestellt werden. Bis jetzt heißt es in einer Anweisung: „Biegen Sie links ab auf die ....!“. Durch die Nutzung von Landmarken könnte z.B. folgendes erreicht werden: „Biegen sie nach der Shell Tankstelle links ab auf die ....!“.

Interessant wäre es auch, die Route nicht nur über eine Karte, sondern zusätzlich noch über ein Video oder ähnliches darzustellen. Beispielsweise könnten von einem OGC *Web 3D Service* eine *Virtual Reality Modeling Language* (VRML) Datei der ermittelten Route erstellt werden. Diese Datei würde einem Nutzer die Route in einer *drei dimensionalen* (3D) Szene präsentieren. Der Nutzer hätte dadurch die Möglichkeit, sich zum Beispiel das Ziel der Route in einem 3D Modell schon vor Reisebeginn anzuschauen.

Der mit meiner Diplomarbeit prototypisch implementierte OpenLS Route Service und die Erweiterungen, wie Emergency Route Service oder die Weboberfläche, können für weitere Entwicklungen oder Verwendungen sehr gut genutzt werden.

Beispielsweise ist mit dem implementierten OpenLS Route Service mehr als nur eine Grundlage für einen Indoor-Routenplaner geschaffen worden. Mit einem gut durchdachten Graphen, der mehrere Stockwerke repräsentieren kann, müsste es möglich sein, den prototypisch implementierten Route Service ohne größere Änderungen direkt wieder verwenden zu können. Somit könnte ein Emergency Route Service für ein Gebäude erstellt werden. Anstatt gesperrten Bereichen oder Straßen, würde es dann dort gesperrte Zimmer, Flure oder Treppenhäuser geben. Es muss aber nicht immer von einem Notfall ausgegangen werden. Der implementierte Route Service könnte auch zur Routenplanung in einem

[1] Traffic Message Channel (TMC)
[2] Mehr Informationen unter: http://www.irt.de/IRT/produkte/IRT_RDS_TMC_Testempfaenger_d.pdf

Gebäude genutzt werden, damit z.B. eine fremde Person ein bestimmtes Zimmer finden kann.

Weil im prototypisch implementierten OpenLS Route Service fast alle *Core Services* (Kapitel 2.2, außer Gateway Service) intern erstellt und verwendet werden, könnten mit diesen Grundlagen aus dem implementierten Route Service auch die Erstellung der restlichen *OpenLS Core Services* erfolgen.

Durch die erstellten Java-Klassen sollte es auch möglich sein, ein PlugIn für OpenJUMP zu entwickeln, welches die Funktion eines Routenplaners im Programm bereitstellt. Der Graph, der zum Routing benötigt wird, muss nicht aus einer Datenbank kommen. Mit kleinen Änderungen könnte er auch aus einem Shapefile erstellt werden.

Die Verbesserungen am ERS (Kapitel 5.2) und an der Weboberläche (Kapitel 5.3) wurden bereits in deren Kapitel teilweise aufgezeigt. Gerade an dem von mir entwickelten ERS sollten noch verschiedene Optimierungen erfolgen. Wie z.B. das angesprochene vorherige eingrenzen der Gefahrenbereiche und Straßen über eine BoundingeBox des Start- und Zielpunktes (ggf. noch Via-Punkte). Bei der Eingrenzung könnte zusätzlich noch die Zeit eine Rolle spielen. Es muss auch nicht die prototypisch erstellte Datenbank verwendet werden, stattdessen könnte auch ein *Web Feature Service* (WFS) die Daten der gesperrten Bereiche und Straßen liefern.

Literaturverzeichnis

[1] APACHE SOFTWARE FOUNDATION: *Apache Tomcat*.
Online unter: http://tomcat.apache.org.

[2] BILL, RALF: *Grundlagen der Geoinformationssysteme*, 1994.

[3] COVER PAGES: *OGC Releases*
OpenGIS Location Services (OpenLS) Implementation Specification.
Online unter: http://xml.coverpages.org/ni2003-04-22-a.html, April 2003.

[4] COWARD & YOSHIDA: *Java Servlet Specification Version 2.4*.
Online unter: http://jcp.org/aboutJava/communityprocess/final/jsr154/index.html.

[5] DIJKSTRA, E. W.: *A note on two problems in connexion with graphs.*
In: Numerische Mathematik. 1 (1959), S. 269271. 1959.

[6] ECLIPSE.
Online unter: http://www.eclipse.org.

[7] EPSG: *European Petroleum Survey Group*.
Online unter: http://www.epsg.org
(abgerufen am 24.04.2006).

[8] FRIDA: *Freie Vektor-Geodaten Osnabrück*.
Online unter: http://frida.intevation.org
(abgerufen am 10.04.2006).

[9] GEOSERVER: *Version 1.3.0*.
Online unter: http://docs.codehaus.org/display/GEOS/Home
(abgerufen am: 14.04.2006).

[10] GEOTOOLS: *The open source Java GIS Toolkit*.
Online unter: http://geotools.codehaus.org.

[11] HTTP: *Hypertext Transfer Protocol - HTTP/1.1*.
Online unter: http://www.ietf.org/rfc/rfc2616.txt, 1999.

[12] JECKLE, M.: *Web Services*.
Online unter: http://www.jeckle.de/webServices.

[13] JON ELLIS, LINDA HO & MAYDENE FISHER: *JDB 3.0 Specification*.
Online unter: http://java.sun.com/products/jdbc/download.html, 2001.

[14] JTS TOPOLOGY SUITE.
Online unter: http://www.vividsolutions.com/JTS/JTSHome.htm.

[15] JUMP PILOT PROJECT: *Offiziele Website vom JUMP Pilot Project (JPP)*.
Online unter: http://jump-pilot.sourceforge.net.

[16] LIF: *Location Interoberability Forum*.
http://www.locationforum.org.

[17] MLP: *Mobile Location Protocol Specification. Verison 3.0*.
Online unter: http://www.openmobilealliance.org/tech/affiliates/lif/lifindex.html
(abgerufen am 06.07.2006).

[18] OGC: *Open Geospatial Consortium*.
http://www.opengeospatial.org.

[19] OPEN GIS CONSORTIUM INC.:
Geography Markup Language Implementation Specification Version 3.1.0.
Online unter: http://www.opengeospatial.org/specs.

[20] OPEN GIS CONSORTIUM INC.: *Simple Features Specification For SQL*.
Online unter: http://www.opengeospatial.org/specs, 1999.

[21] OPEN GIS CONSORTIUM INC.: *Filter Encoding Implementation Specification*.
Online unter: http://www.opengeospatial.org/specs, 2001.

[22] OPEN GIS CONSORTIUM INC.:
Styled Layer Descriptor Implementation Specification Version 1.0.0.
Online unter: http://www.opengeospatial.org/specs, 2002.

[23] OPEN GIS CONSORTIUM INC.:
Web Feature Service Implementation Specification Version 1.0.
Online unter: http://www.opengeospatial.org/specs, 2002.

[24] OPEN GIS CONSORTIUM INC.:
Web Map Service Implementation Specification Version 1.1.1.
Online unter: http://www.opengeospatial.org/specs, 2002.

[25] OPENLS: *OGC Open Location Services Version 1.1*.
Online unter: http://www.opengeospatial.org/functional/?page=ols.

[26] POSTGIS.
Online unter: http://www.postgis.org.

[27] POSTGRESQL: *Das freie objektrelationale Open Source Datenbanksystem*.
Deutsche Projekt Seite online unter: http://www.postgres.de.

[28] POSTGRESQL: *The world's most advanced open source database*.
Online unter: http://www.postgresql.org.

[29] SOURCEFORGE.NET: *The JUMP Pilot Project*.
Online unter: http://sourceforge.net/projects/jump-pilot
(abgerufen am 22.02.2006).

[30] SUN: *Sun Microsystems GmbH*.
Online unter: http://de.sun.com.

[31] VIVID SOLUTIONS: *Unternehmen welches die original Version von JUMP erstellte*.
Online unter: http://www.vividsolutions.com.

[32] W3C: *World Wide Web Consortium*.
Online unter: http://www.w3.org.

[33] W3C: *Working Group - Web Services Glossary*.
Online unter: http://www.w3.org/TR/ws-gloss, 2004.

[34] WIKIPEDIA: *Die frei Enzyklopädie*.
http://de.wikipedia.org/wiki/Hauptseite.

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

A Anhang

A.1 GetMap Request Geoserver WMS

Folgend ein Beispiel für einen GetMap Request an den Geoserver WMS via HTTP-POST. (Wichtig, nicht jeder WMS unterstützt HTTP-POST und SLD!).

```
<GetMap version="1.2.0" service="WMS"
xsi:schemaLocation="http://www.opengis.net/ows D:sld/GetMap.xsd"
xmlns="http://www.opengis.net/ows"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:sld="http://www.opengis.net/sld"
xmlns:gml="http://www.opengis.net/gml">
  <sld:StyledLayerDescriptor version="1.0.0">
    <sld:NamedLayer>
      <sld:Name>topp:gruenflaechen-joined</sld:Name>
      <sld:NamedStyle><sld:Name>gruenflaechen</sld:Name></sld:NamedStyle>
    </sld:NamedLayer>
    <sld:NamedLayer>
      <sld:Name>topp:gewaesserflaechen-joined</sld:Name>
      <sld:NamedStyle><sld:Name>gewaesserflaechen</sld:Name></sld:NamedStyle>
    </sld:NamedLayer>
    <sld:NamedLayer>
      <sld:Name>topp:gewaesserlinien-joined</sld:Name>
      <sld:NamedStyle><sld:Name>gewaesserlinien</sld:Name></sld:NamedStyle>
    </sld:NamedLayer>
    <sld:NamedLayer>
      <sld:Name>topp:strassen-joined</sld:Name>
      <sld:NamedStyle><sld:Name>strassen</sld:Name></sld:NamedStyle>
    </sld:NamedLayer>
    <sld:NamedLayer>
      <sld:Name>topp:poi-joined</sld:Name>
      <sld:NamedStyle><sld:Name>pois</sld:Name></sld:NamedStyle>
    </sld:NamedLayer>
    <sld:UserLayer>
      <sld:Name>Route</sld:Name>
      <sld:InlineFeature>
```

```
        <Route xmlns="">
          <the_geom>
            <LineString srsName="31467" xmlns="http://www.opengis.net/gml">
              <coord><X>3433844.000</X><Y>5796218.150</Y></coord>
              <coord><X>3433826.180</X><Y>5796211.340</Y></coord>
              <coord><X>3433808.370</X><Y>5796204.530</Y></coord>
            </LineString>
          </the_geom>
        </Route>
      </sld:InlineFeature>
      <sld:LayerFeatureConstraints><sld:FeatureTypeConstraint/>
      </sld:LayerFeatureConstraints>
      <sld:UserStyle>
        <sld:FeatureTypeStyle>
          <sld:Rule>
            <sld:LineSymbolizer>
              <sld:Stroke>
                <sld:CssParameter name="stroke">#990000</sld:CssParameter>
                <sld:CssParameter name="stroke-width">4</sld:CssParameter>
              </sld:Stroke>
            </sld:LineSymbolizer>
          </sld:Rule>
        </sld:FeatureTypeStyle>
      </sld:UserStyle>
    </sld:UserLayer>
    <sld:UserLayer>
      <sld:Name>START</sld:Name>
      <sld:InlineFeature>
        <Point xmlns="">
          <the_geom>
            <Point srsName="31467" xmlns="http://www.opengis.net/gml">
              <coord><X>3433844</X><Y>5796218.15</Y></coord>
            </Point>
          </the_geom>
          <name>START</name>
        </Point>
      </sld:InlineFeature>
      <sld:LayerFeatureConstraints>
        <sld:FeatureTypeConstraint/>
      </sld:LayerFeatureConstraints>
      <sld:UserStyle>
        <sld:FeatureTypeStyle>
          <sld:Rule>
            <sld:Name>Rule</sld:Name>
            <sld:Title>START</sld:Title>
```

```
76            <sld:PointSymbolizer>
77              <sld:Graphic>
78                <sld:Mark>
79                  <sld:WellKnownName>circle</sld:WellKnownName>
80                  <sld:Fill>
81                        <sld:CssParameter name="fill">#990000</sld:CssParameter>
82                  </sld:Fill>
83                </sld:Mark>
84                <sld:Size>8.0</sld:Size>
85              </sld:Graphic>
86            </sld:PointSymbolizer>
87            <sld:TextSymbolizer>
88              <sld:Label>
89                <ogc:PropertyName xmlns:ogc="http://www.opengis.net/ogc">
90                        name</ogc:PropertyName>
91              </sld:Label>
92              <sld:Font>
93                <sld:CssParameter name="font-family">Arial</sld:CssParameter>
94                <sld:CssParameter name="font-style">Bold</sld:CssParameter>
95                <sld:CssParameter name="font-size">12</sld:CssParameter>
96              </sld:Font>
97              <sld:LabelPlacement>
98                <sld:PointPlacement>
99                  <sld:Displacement>
100                   <sld:DisplacementX>6</sld:DisplacementX>
101                   <sld:DisplacementY>0</sld:DisplacementY>
102                 </sld:Displacement>
103               </sld:PointPlacement>
104             </sld:LabelPlacement>
105             <sld:Fill>
106               <sld:CssParameter name="fill">#990000</sld:CssParameter>
107             </sld:Fill>
108           </sld:TextSymbolizer>
109         </sld:Rule>
110       </sld:FeatureTypeStyle>
111     </sld:UserStyle>
112   </sld:UserLayer>
113   <sld:UserLayer>
114     <sld:Name>END</sld:Name>
115     <sld:InlineFeature>
116       <Point xmlns="">
117         <the_geom>
118           <Point srsName="31467" xmlns="http://www.opengis.net/gml">
119             <coord><X>3433808.37</X><Y>5796204.53</Y></coord>
120           </Point>
```

```
121           </the_geom>
122           <name>END</name>
123         </Point>
124       </sld:InlineFeature>
125       <sld:LayerFeatureConstraints>
126         <sld:FeatureTypeConstraint/>
127       </sld:LayerFeatureConstraints>
128       <sld:UserStyle>
129         <sld:FeatureTypeStyle>
130           <sld:Rule>
131             <sld:Name>Rule</sld:Name>
132             <sld:Title>END</sld:Title>
133             <sld:PointSymbolizer>
134               <sld:Graphic>
135                 <sld:ExternalGraphic>
136                   <sld:OnlineResource xlink:href=
137                         "http://localhost:8080/geoserver/data/symbols/endflag.gif"
138                         xlink:type="simple"
139                         xmlns:xlink="http://www.w3.org/1999/xlink"/>
140                   <sld:Format>image/gif</sld:Format>
141                 </sld:ExternalGraphic>
142                 <sld:Size>30</sld:Size>
143               </sld:Graphic>
144             </sld:PointSymbolizer>
145             <sld:TextSymbolizer>
146               <sld:Label>
147                 <ogc:PropertyName xmlns:ogc="http://www.opengis.net/ogc">
148                         name</ogc:PropertyName>
149               </sld:Label>
150               <sld:Font>
151                 <sld:CssParameter name="font-family">Arial</sld:CssParameter>
152                 <sld:CssParameter name="font-style">Bold</sld:CssParameter>
153                 <sld:CssParameter name="font-size">12</sld:CssParameter>
154               </sld:Font>
155               <sld:LabelPlacement>
156                 <sld:PointPlacement>
157                   <sld:Displacement>
158                     <sld:DisplacementX>6</sld:DisplacementX>
159                     <sld:DisplacementY>-5</sld:DisplacementY>
160                   </sld:Displacement>
161                 </sld:PointPlacement>
162               </sld:LabelPlacement>
163               <sld:Fill>
164                 <sld:CssParameter name="fill">#990000</sld:CssParameter>
165               </sld:Fill>
```

```
166             </sld:TextSymbolizer>
167           </sld:Rule>
168         </sld:FeatureTypeStyle>
169       </sld:UserStyle>
170     </sld:UserLayer>
171   </sld:StyledLayerDescriptor>
172   <BoundingBox srsName="http://www.opengis.net/gml/srs/epsg.xml#31463">
173     <gml:coord><gml:X>3433708.37</gml:X><gml:Y>5796093.52</gml:Y></gml:coord>
174     <gml:coord><gml:X>3433944</gml:X><gml:Y>5796329.15</gml:Y></gml:coord>
175   </BoundingBox>
176   <Output>
177     <Format>image/png</Format>
178     <Transparent>false</Transparent>
179     <BGcolor>#FFFFFF</BGcolor>
180     <Size>
181         <Width>400</Width>
182         <Height>400</Height>
183     </Size>
184   </Output>
185   <Exceptions>application/vnd.ogc.se+xml</Exceptions>
186 </GetMap>
```

Listing A.1: GetMap Request an Geoserver WMS

A.2 UML-Klassendiagramme

Die folgende Abbildung A.1 ist eine Übersicht der erstellten *packages* incl. deren Klassen und Schnittstellen des implementierten Route Services (verschiedene Klassen wurden aus Gründen der Übersicht aus diversen *packages* weggelassen). Sie zeigt zusätzlich noch die vereinfachte Benutzung untereinander.

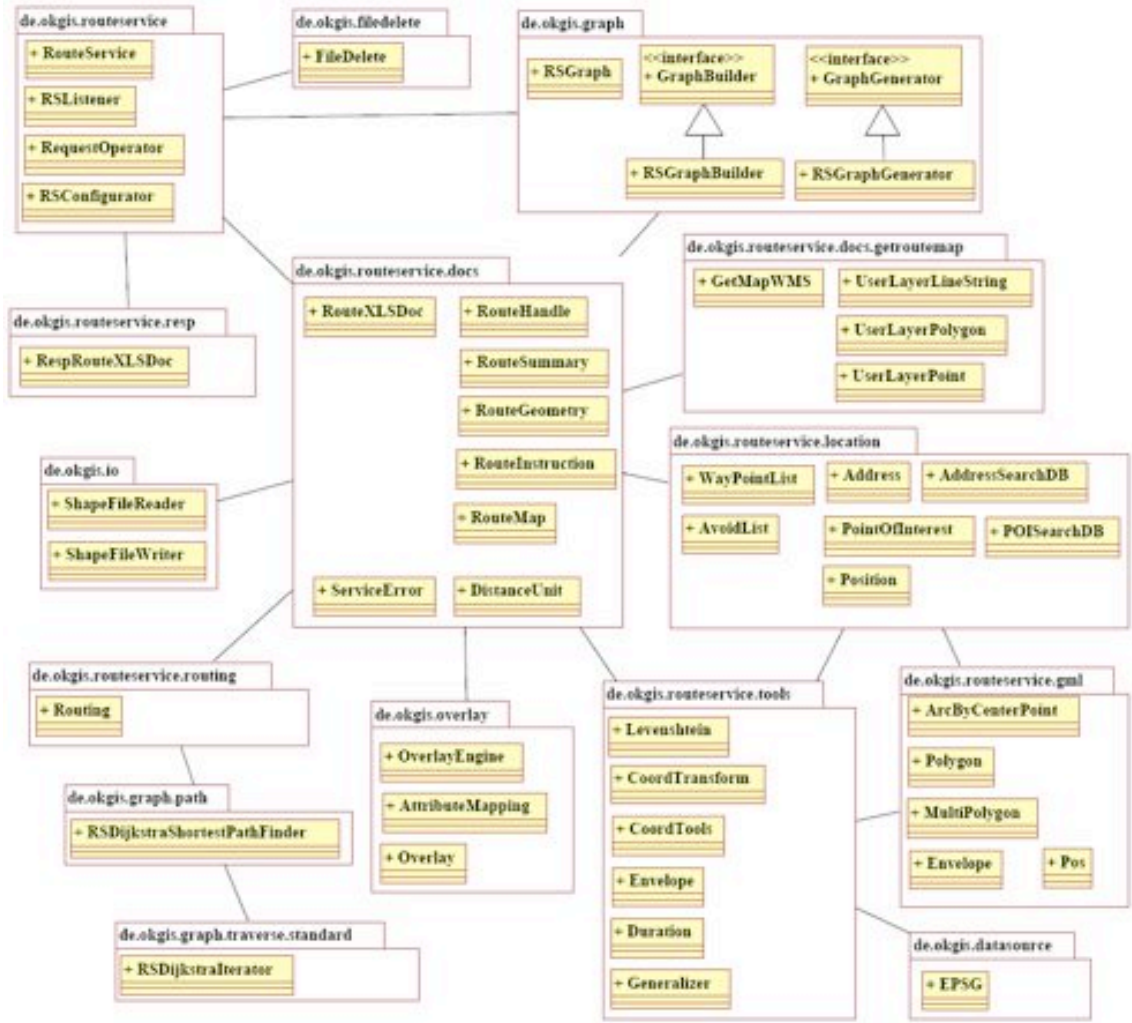


Abbildung A.1: UML-Klassendiagramm Übersicht

Auf der beiliegenden CD befinden sich neben einer Java-Doc noch die UML-Klassendiagramme für alle in der Abbildung A.1 dargestellten *packages* incl. deren Methoden.

UML-Klassendiagramme des Emergency Route Service und der Weboberfläche befinden sich ebenfalls auf der beiliegenden CD.

A.3 Inhaltsverzeichnis CD

Die CD ist wie folgt aufgebaut:

1. Testdaten Frida Osnabrück - enthält drei Ordner:
- Datenbanken Daten - (Tabellen: OSNA, OSNA_POI, AOIToAvoid, AddressToAvoid)
- Geoserver Daten - (Shapefiles, SLDs, Symbole)
- Geoserver Dokumentation - (Dokumentation als gepackte Datei)

2. Tomcat WAR Dateien - enthält drei Ordner:
- Emergency Route Service
- OpenLS Route Service
- Weboberfläche für RS und ERS

3. Java-packages - enthält fünf Ordner:
- Emergency Route Service incl. Java-Doc, Klassendiagramme
- Route Service incl. Java-Doc, Klassendiagramme
- Tool EdgeIDToPoint
- Verwendete GeoTools Java Archive beim OpenLS Route Service
- Weboberfläche für RS und ERS incl. Java-Doc, Klassendiagramme

4. OpenLS Spezifikation Version 1.1 - enthält:
- OpenLS Spezifikation als PDF-Dokument
- Ausführliche OpenLS Schema Dokumentation als PDF- und HTML-Dokument

5. Diplomarbeit als PDF-Dokument

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

....

(Ort, Datum) (Unterschrift)