GEORG-SIMON-OHM
HOCHSCHULE NÜRNBERG

Fakultät Elektrotechnik Feinwerktechnik Informationstechnik

Studiengang Elektro- und Informationstechnik
Vertiefungsrichtung Informationstechnik

Diplomarbeit

von **Stefan Scholz** mit dem Titel

Entwicklung eines Tools zur Konvertierung von OpenStreetMap-Kartendaten für die Navigation

Matrikelnummer:	2039056
Semester:	WS 2011
Erstgutachter:	Prof. Dr. Hans-Georg Hopf
Zweitgutachter:	Prof. Dr. Helmut Herold
Abgabedatum:	31. Oktober 2011
Firma:	Elektrobit Automotive
Abteilung:	NAVI-4
Betreuer:	Dr. Sebastian Fischer

Schlagworte: OpenStreetMap, GDF, Navigation, Spatialite, SQLite, OpenGIS

Erklärung gemäß RaPO §35 (7)

Ich bestätige hiermit, dass ich meine Diplomarbeit mit dem Titel:

Entwicklung eines Tools zur Konvertierung von OpenStreetMap-Kartendaten für die Navigation

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Nürnberg, den 26. Oktober 2011

STEFAN SCHOLZ

Inhaltsverzeichnis

Abkürzungsverzeichnis **IV**

Abbildungsverzeichnis **VI**

Tabellenverzeichnis **VIII**

Verzeichnis der Listings **IX**

1. Einleitung **1**
1.1. Hinführung zum Thema 1
1.2. Ziel der Arbeit 2
1.3. Vorgehensweise 3
1.4. Gliederung der Arbeit 3

2. OpenStreetMap **4**
2.1. Grundlagen 5
2.2. Datenquellen und Mapping-Praxis 8
2.3. OpenStreetMap - API 13
2.4. Lizenz 17
2.5. Anwendungen 18
2.6. Datenqualität 22

3. Elektrobit Automotive **24**
3.1. EB street director 24
3.2. Kartenübersetzungsprozess 29
3.3. Physical Storage Format (PSF) 30

4. Geographic Data Files **32**
4.1. Datenmodell 32
4.2. Dateiformat 35

5. Spatialite **37**
5.1. SQLite . . . 37
5.2. OpenGIS . . . 39
5.3. Aufbau von Spatialite . . . 44
5.4. Besonderheiten bei der Verwendung . . . 47

6. Bearbeitung der Aufgabenstellung **51**
6.1. Konzept . . . 51
6.2. Vorgehensweise . . . 53
6.3. Übersicht . . . 54
6.4. Implementierung . . . 56
6.4.1. Erweiterung des GDFReaders . . . 57
6.4.2. Einlesen der OSM-Daten . . . 58
6.4.3. Klassenmodell für die OSM-Daten . . . 59
6.4.4. Datenmodell für die OSM-Daten . . . 60
6.4.5. Cursor-Konzept . . . 61
6.4.6. Vorverarbeitung der OSM-Daten . . . 63
6.4.7. Ermittlung des FeatureClassCodes . . . 67
6.4.8. Erzeugung der GDF-Attribute . . . 70
6.4.9. Datenmodell für den Export der GDF-Elemente . . . 72
6.4.10. Export der grafischen GDF-Elemenete . . . 75
6.4.11. Erzeugung von Informationen für das Geocoding . . . 82
6.4.12. Realisierung einer einfachen Konfiguration . . . 88
6.5. Verwendete Algorithmen . . . 88
6.5.1. Berechnung des Uhrzeigersinns . . . 89
6.5.2. Punkt-In-Polygon . . . 91
6.5.3. Multipolygon . . . 91
6.6. Ergebnisse . . . 92

7. Zusammenfassung und Ausblick **95**

Literaturverzeichnis **98**

A. Anhang **i**
A.1. Verwendete Tools . . . ii
A.1.1. Entwicklung . . . ii
A.1.2. OpenStreetMap . . . iii
A.1.3. Elektrobit . . . iv

A.1.4. Dokumentation v
A.1.5. Andere nützliche Tools vi
A.2. Glossar vii

Abkürzungsverzeichnis

ADAS ..... Advanced driver assistance systems
BLOB ..... Binary Large Object
CLI ..... Command Line Interface
DCL ..... Data Control Language
DOM ..... Document Object Model
DTM ..... Digital Terrain Model
EB ..... Elektrobit
FCC ..... Feature Class Code
GDF ..... Geographic Data File
GEOS ..... Geometry Engine - Open Source
GIS ..... Geografisches Informationssystem
GPS ..... Global Positioning System
HMI ..... Human-Machine-Interface
HTTP ..... Hypertext Transfer Protocol
MapCo ..... Map-Compilation
OGC ..... Open Geospatial Consortium
OSM ..... Open-Street-Map
PND ..... Portable Navigation Device
POI ..... Point of Interest
PSF ..... Physical Storage Format
SAX ..... Simle API for XML
SDK ..... Software Development Kit
TTS ..... Text To Speech
URI ..... Uniform Resource Identifier
WGS 84 ..... World Geodetic System 1984
WKB ..... Well-Known Binary

WKT ..... Well-Known Text
WMS ..... Web Map Service
XML ..... eXtensible Markup Language

Abbildungsverzeichnis

2.1. OSM - Hauptkarte 7
2.2. JOSM - Ebenen und Tags 10
2.3. JOSM - Georeferenzierte Fotos 11
2.4. JOSM - Erstellung eines Multipolygons 12
2.5. Empfangsbereiche für digitales Radio 19
2.6. Abweichungen von der Position kleiner 10 m 22
2.7. Länge des OSM-Straßennetzes im Vergleich zu TeleAtlas 23

3.1. EB street director Application 24
3.2. Elektrobit StreetDirector 25
3.3. EB Guide 26
3.4. Kartendarstellungen 27
3.5. Tools für den **EB street director** 28
3.6. Kartenübersetzungsprozess bei EB 29
3.7. Aufbau des PSF-Formats 30

4.1. GDF Datenmodell 33

5.1. Architektur von SQLite 38
5.2. OpenGIS-Datentypen (vereinfacht) 40
5.3. Aufbau von Spatialite 44
5.4. Ergebnis der QueryPlan - Abfrage 48
5.5. Linien und Flächen in Spatialite-GIS 49

6.1. Konzept für den zu entwickelnden Konverter 52
6.2. Konzept für den OSMReader 52
6.3. Übersicht zum entwickelten Konverter 54
6.4. Datenmodell 56
6.5. Erweiterung des GDFReaders 57
6.6. XML-Import 58
6.7. OSM Basisklassen 59

6.8. Datenmodell für den OSM-Import 60
6.9. Cursor für die OSM-Grundelemente 61
6.10. OsmPostProcessor 63
6.11. OsmPostProcessor - Klassen 63
6.12. Kennzeichnung von Kreuzungen und Wegenden 64
6.13. Punkt aus Polygon 65
6.14. Datenstruktur zur Ermittlung des FeatureClassCodes 67
6.15. Klassendiagramm FeatureClassResolver 69
6.16. Attribute im GDF-Format 70
6.17. Ermittlung der Attribute 71
6.18. Datenmodell für den GDF-Export 73
6.19. Gemeinsamkeiten mit dem GDF Datenmodell 74
6.20. vereinfachtes GDF Datenmodell 75
6.21. Datenstrukturen aus den Grundelementen 76
6.22. Schnittstelle zur Datenbank 78
6.23. Übersicht GdfExport 78
6.24. LineFeatureExport 79
6.25. ComplexAreaExport 81
6.26. Geocoding in GDF (Deutschland) 82
6.27. vereinfachtes Geocoding 83
6.28. GeoModel 84
6.29. Order8-Grenzen in Spatialite-GIS 85
6.30. Straßen innerhalb eines Order8 85
6.31. BuiltUp und Place in Spatialite-GIS 86
6.32. ExportGeocoding 87
6.33. Konfiguration 88
6.34. Beispiel für zwei Vektoren im Uhrzeigersinn 89
6.35. Punkt-In-Polygon nach Jordan 91
6.36. Generierte Karte im MapViewer 92
6.37. Zieleingabe im EB street director 93
6.38. Zielführung im EB street director 94

Tabellenverzeichnis

2.1. Beispiele für MapFeatures 6
2.2. Attribute für Restaurant 7
2.3. Beispiele für Relationen 8
2.4. OpenStreetMap-API für Datenelemente 14

5.1. Well Known Text - Format 41
5.2. Auszug aus den, in OpenGIS definierten, SQL - Funktionen 42
5.3. zusätzliche SQL-Funktionen von Spatialite 43

6.1. Relationen für das Geocoding (Ausschnitt) 84

Verzeichnis der Listings

2.1. XML-Darstellung einer Node 15
2.2. XML-Darstellung eines Ways 15
2.3. XML-Darstellung einer Relation 16

4.1. GDF - Textformat 35

5.1. Schnittstelle von `sqlite3_create_function` 44
5.2. Hinzufügen der Funktion `ABS` zum SQLite-Befehlssatz 45
5.3. Query Plan 48
5.4. geometrische Abfrage ohne Index 49
5.5. geometrische Abfrage mit Index 50
5.6. Erzeugen eines räumlichen Index 50
5.7. Von *CreateSpatialIndex* erzeugter INSERT-Trigger 50

6.1. Verwendung von `OsmNodeCursor` 62
6.2. Markierung von Kreuzungen mit dem `OsmNodeCursor` 64
6.3. Punkt aus Polygon 66
6.4. Initialisierung des FccResolvers 68
6.5. Übersetzung der Attribute für eine Straße (Auszug) 71
6.6. Export eines einfachen LineFeatures 77
6.7. Verwendung der Konfiguration 88
6.8. Berechnung des Uhrzeigersinns 90

1. Einleitung

1.1. Hinführung zum Thema

Die Verwendung eines Navigationsgerätes im Auto ist heute für viele eine Selbstverständlichkeit. Dies ist nicht schwer nachzuvollziehen, denn kaum eine andere Technologie erhöht den Fahrkomfort in einer Weise, wie es eine verlässliche und einfach zu bedienende Navigationslösung vermag. Gehörte vor nicht allzulanger Zeit ein Autoatlas zur Standardausstattung eines jeden Kraftfahrzeugs, so wurde dieser heute fast vollkommen von den neuen Geräten ersetzt. Dabei sind deren Daten um vieles detaillierter und oft auch deutlich aktueller als die meisten Kartenwerke. Vorbei ist die Zeit, in der die Fahrt in den Urlaub oder in eine fremde Stadt noch von langer Hand geplant werden musste. Der moderne Mensch steigt in sein Fahrzeug, wählt ein gewünschtes Reiseziel und wird kurz darauf von einer freundlichen Stimme begleitet, die ihn sicher an den gewünschten Ort führt. Was im Auto bereits so selbstverständlich geworden ist, scheint aber in einigen Bereichen noch undenkbar. So werden beim Wandern, bei Fahrradtouren oder Trekkingreisen noch vorrangig Kartenwerke benutzt. Auch als Tourist in einer fremden Stadt gehört der Stadtplan noch zur Standardausrüstung. Dies ist etwas verwunderlich da leistungsstarke Hardware, für die Hosentasche, bereits zu günstigen Preisen erhältlich ist. Dabei mangelt es auch nicht an der Möglichkeit einer Positionsbestimmung oder ausreichender Akkulaufzeit.

Der Grund hierfür befindet sich an einer anderen Stelle. Wer schon einmal versucht hat ein, fürs Auto konzipiertes, Navigationgerät als Fußgänger oder Radfahrer zu verwenden, wird selten von den Ergebnissen begeistert sein. So müssen häufig Umwege in Kauf genommen werden, da Wege, die nicht mit dem Auto befahrbar sind, dem Navigationsgerät gänzlich unbekannt sind. Es gibt, mit NAVTEQ (Nokia) und Teleatlas (TomTom), weltweit nur zwei große kommerzielle Anbieter von Navigationsdaten. Diese werden, fast ausnahmslos, für alle kommerziellen Navigationslösungen verwendet. Auch das Kartenangebot von Google entstammt größtenteils den Daten eines dieser Anbieter. Damit diese, mit vertretbarem Aufwand, ständig aktuelle Kartendaten

liefern können, werden Fahrzeuge eingesetzt, die die Erfassung der Daten weitestgehend automatisieren. Die Erfassung von Fuß- oder Radwegen ist auf diese Weise nicht möglich, wodurch deren Fehlen erklärt werden kann.

Das OpenStreetMap-Projekt löst dieses Problem auf eine sehr elegante Art und Weise. Hierzu wird vergleichsweise einfache Technik eingesetzt um Kartendaten zu erfassen. Dafür setzt das Projekt auf eine große Anzahl an freiwilligen Datensammlern. Diese verwenden einfache GPS-Geräte um den Verlauf von Straßen und Wegen zu erfassen oder zeichnen Gebäude, Wälder oder Seen von Luftaufnahmen ab. Mit Hilfe dieser Vorgehensweise konnte OpenStreetMap innerhalb weniger Jahre Erstaunliches erreichen. So sind die Daten, in einigen Bereichen, bereits viel detaillierter als die der kommerziellen Datenlieferanten und enthalten Informationen, die eine Navigation auch abseits von Straßen ermöglicht. Zudem stehen diese Daten für Jedermann kostenlos zur Verfügung.

Es gibt zwar schon einige interessante Navigationslösungen auf der Basis von OpenStreetMap-Daten, jedoch können diese bis jetzt nur einige technikbegeisterte Anwender ansprechen. Betrachtet man Downloadstatistiken und Bewertungen von Navigationlösungen, für aktuelle Smartphone-Plattformen, so scheint, für einen Großteil der Benutzer, einfache Bedienbarkeit und eine schöne Kartendarstellung einen höheren Stellenwert zu besitzen als die Detailtreue der verwendeten Daten. Ungeachtet des oft hohen Preises, sind die am häufigsten installierten Navigationsprogramme meist Angebote kommerzieller Hersteller.

Die Grundidee für diese Arbeit ist, die Vorzüge der frei erhältlichen OpenStreetMap-Daten mit denen des **EB street director**, einer ausgereiften kommerziellen Navigationslösung der Firma Elektrobit, zu vereinen.

1.2. Ziel der Arbeit

Ziel dieser Arbeit ist der Entwurf eines Konverters, der es ermöglicht OpenStreetMap-Kartendaten im **EB street director** zu nutzen. Dabei soll sich die Lösung einfach in den bestehenden Kartenübersetzungsprozess der Firma Elektrobit integrieren lassen.

1.3. Vorgehensweise

Es ist geplant, das Datenmodell des, in der Navigationsbranche verbreiteten, GDF-Formats in einer relationalen Datenbank nachzubilden und mit OpenStreetMap-Daten zu füllen. Dazu soll Spatialite verwendet werden, eine Erweiterung der bekannten SQLite-Datenbank um OpenGIS-kompatible Geometriedatentypen. Zur weiteren Verarbeitung muss der Kartenübersetzungsprozess, der Firma Elektrobit, um einen geeigneten Import, der so gewonnenen Daten, erweitert werden.

1.4. Gliederung der Arbeit

Nach einer Vorstellung des OpenStreetMap-Projekts, die neben der verwendeten Technik auch auf einige Aspekte wie die verwendete Lizenz oder die Datenqualität eingeht, werden die Eigenschaften des **EB street director**, der grundsätzliche Ablauf der Kartenübersetzung bei Elektrobit und das firmeneigene PSF-Format beschrieben. Auf eine grundlegende Einführung in das GDF-Format folgt eine Beschreibung der Eigenschaften der, in diesem Projekt verwendeten, Spatialite-Datenbank. Nach einem Kapitel, das die Lösung der gestellten Aufgabe skizziert und die erzielten Ergebnisse beschreibt, folgt eine Zusammenfassung mit Ausblick.

2. OpenStreetMap

OpenStreetMap (OSM) wurde im Juli 2004 von Steve Coast gegründet, mit dem Ziel eine freie Weltkarte zu schaffen. Das Projekt dient der Erfassung und Nutzung freier Geoinformationen, die größtenteils durch Hobby-Kartografen zusammengetragen werden. Mit Hilfe von GPS-Geräten und dafür freigegebenen Luftaufnahmen zeichnen diese die Welt in einem einfachen vektorbasierten Format nach, welches aus nur drei Grundelementen besteht. Trotz des einfachen Formates kann aber mit OpenStreetMap ein viel höherer Detailierungsgrad, als für kommerzielle Karten üblich, erreicht werden. Gezeichnet werden kann nämlich grundsätzlich alles, was der Ersteller der Karte (der sogenannte **Mapper**) für relevant hält. Auf der Wiki-Seite des Projekts findet hierzu ein reger Austausch über neue Kartenelemente und deren Kennzeichnung statt [1]. Im Jahr 2006 wurde die OpenStreetMap-Foundation gegründet, die sich aus Spendengeldern finanziert und die Server betreibt. Seitdem ist auch eine Infrastruktur vorhanden um große Gebiete zu kartografieren. Ebenso wuchs das Interesse an der freien Karte und die Anzahl der registrierten Benutzer stieg bis zum heutigen Tag auf etwas über 400 000 [2]. Zur Arbeit der ehrenamtlichen Helfer kommen viele Datenspenden aus unterschiedlichsten Quellen. Schon heute sind OpenStreetMap-Karten vor allem in großen Städten um vieles detaillierter als kommerzielle Karten und das Erreichen einer weltweiten Abdeckung nur eine Frage der Zeit. Wegen der ähnlichen Vorgehensweise beim Sammeln von Informationen wird OpenStreetMap gerne mit der freien Enzyklopädie Wikipedia verglichen und auch als Wiki-Weltkarte bezeichnet. Natürlich gibt es auch Kritik an dieser Vorgehensweise. Diese wird bei OpenStreetMap oft durch die Verwendung von einfachen GPS-Geräten, welche die aktuelle Position nur auf etwa 5 Meter genau bestimmen können, begründet. Für eine Vielzahl von Anwendungen sind die Daten jedoch genau genug [3].

In diesem Kapitel werden zunächst die Grundlagen und Begrifflichkeiten von OpenStreetMap vorgestellt, worauf eine Einführung in die Arbeitsweise und die verwendeten

[1] `http://wiki.openstreetmap.org/wiki/Map_Features`
[2] `http://wiki.openstreetmap.org/wiki/Stats`
[3] [Topf, 2010]

Werkzeuge der Mapper folgt. Nach einer Beschreibung der zugrundeliegenden Technik und dem Aufbau des Datenformats wird die verwendete Lizenz kurz beschrieben und einige Anwendungsgebiete für die Daten vorgestellt. Das Kapitel schließt mit einer kurzen Betrachtung zur Qualität der OSM-Daten.

2.1. Grundlagen

In OpenStreetMap gibt es drei Grundelemente. Jedoch besteht ein Großteil der OSM-Daten aus nur zweien von diesen:

- **Nodes**
  (Punkte oder Knoten) stellen eine Position dar und haben eine geometrische Länge und Breite
- **Ways**
  (Wege) verbinden mehrere Nodes zu einer Linie und können auch als Flächen interpretiert werden, wenn diese Linie einen Ring bildet

Alle Elemente in OSM können mit Attributen, den sogenannten **Tags** versehen werden. Ein Tag besteht dabei immer aus zwei, grundsätzlich beliebigen, Zeichenketten von denen die Erste als **Key** (Schlüssel) und die Zweite als **Value** (Wert) bezeichnet wird.

Natürlich wäre eine Karte, in der jeder nach Belieben Attribute vergibt, für die Allgemeinheit nutzlos. Deshalb gibt es im Wiki des Projektes eine lange Liste mit Tags auf die sich die Mitglieder geeinigt haben [4]. Das Konzept ist leicht zu verstehen - aus einem Grundelement mit ensprechendem Tag wird ein sogenanntes **MapFeature**, für das es eine Kartendarstellung gibt (Tabelle 2.1).

Key	Value	Element	Rendering
amenity	post_box		
highway	tertiary		
amenity	restaurant		
natural	wood		
building	yes		

Tabelle 2.1.: Beispiele für MapFeatures

[4] http://wiki.openstreetmap.org/wiki/Map_Features

Für ein einzelnes MapFeature gibt es eine Liste weiterer Tags, mit denen es genauer beschrieben werden kann:

amenity = restaurant	
name	Restaurant Mustermann
cuisine	italian
smoking	no
opening_hours	Mo-Sa 09:00-20:00

Tabelle 2.2.: Attribute für Restaurant

Mit diesen einfachen Mitteln lassen sich schon die meisten Karten zeichnen. Mit dem Node-Element können **Points Of Interest** (POI), wie Restaurants oder Haltestellen und mit dem Way-Element Straßen, Flüsse, Gebäude oder Waldflächen gezeichnet werden.

Abbildung 2.1.: OSM - Hauptkarte

Das letzte Element in OpenStreetMap ist die **Relation** (Zusammenhang). Mit ihr können auch komplexe Zusammenhänge modelliert werden, für die Nodes und Ways alleine nicht ausreichen. Eine Relation hat mindestens ein Tag, das den Typ angibt und kann aus Nodes, Ways und sogar wieder aus Relationen bestehen. Diese werden in diesem Zusammenhang als **Member** (Mitglieder) bezeichnet, denen eine **Role** (Funktion) zugewiesen werden kann. Es gibt sehr viele unterschiedliche Relationstypen, von denen die hier genannten lediglich als Beispiel dienen:

- **Multipolygon**
  dient der Modellierung von zusammengesetzten Flächen. Mit ihm können zum Beispiel Gebäude mit Innenhöfen, Waldlichtungen und Ähnliches dargestellt werden. Ein Multipolygon besteht aus mehreren Ways die, wenn es sich um eine Außenfläche handelt, mit der Role outer versehen werden. Für Innenflächen wird entsprechend inner verwendet. Der Umriss einer Fläche kann dabei auch durch mehrere Ways beschrieben werden, die sich aufgrund ihrer Start- und Endpunkte zu einem Ring verbinden lassen.

- **Restriction**
  kann für Abbiegeverbote verwendet werden. Sie besteht aus zwei Ways und einer Node. Die mit den Rollen `from`, `to` und `via` versehen werden. Das Tag `restriction=no_left_turn` gibt beispielsweise an, dass die angegebene Verbindung wegen eines Verbotes für Linksabbieger gesperrt ist.

- **Route**
  ist ein weiterer nützlicher Relationstyp, mit dem beispielsweise Bus- und Straßenbahnlinien oder auch Fahrrad- und Wanderrouten beschrieben werden können. Diese Relation ist sehr vielseitig und definiert eine große Menge verschiedener Tags und Roles. Im Falle einer Buslinie werden Nodes als Haltestellen und Ways als Verbindungen zwischen diesen verwendet. Das Tag `route=bus` gibt an, dass es sich hierbei um eine Busverbindung handelt.

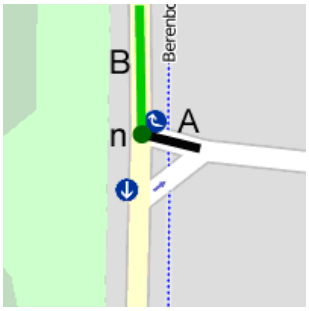
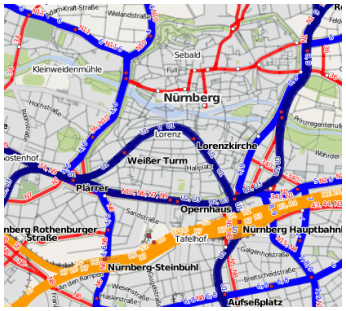
multipolygon	restriction	route
		

Tabelle 2.3.: Beispiele für Relationen

2.2. Datenquellen und Mapping-Praxis

In diesem Abschnitt soll gezeigt werden, welche freien Datenquellen für Geodaten es gibt und wie diese nach OSM übertragen werden können. Hierfür sollen einige

Werkzeuge, die den Mappern zur Verarbeitung der Daten zur Verfügung stehen, am Beispiel des JOSM-Editors vorgestellt werden. Zu den wichtigsten Datenquellen für OpenStreetMap gehören:

- **Satellitenbilder**
  die zum Abzeichnen für OpenStreetMap freigegeben sind. Sie stehen in unterschiedlichen Auflösungen in fast allen Teilen der Welt zur Verfügung. Waldflächen, Flüsse, Seen oder Umrisse von Gebäuden können am Besten auf diesem Wege erfasst werden. Von Satellitenbildern abgezeichnete Straßenverläufe können den Mappern, die ein neues Gebiet erschließen, als erste Orientierung dienen.

- **GPS-Tracks**
  können von einer Vielzahl einfacher GPS-Empfänger erzeugt werden. Der Empfänger speichert hierfür regelmäßig (z. B. einmal in der Sekunde) die aktuelle Position als Wegpunkt ab, was ein Bewegungsprofil ergibt. Aus diesem Profil können dann genaue Verläufe von Straßen oder Wegen abgeleitet werden. Oft gibt es noch die Möglichkeit die momentane Position auf Knopfdruck speziell zu kennzeichnen um beispielsweise einen POI zu markieren. Aufgezeichnete Tracks können hochgeladen und somit auch anderen Mappern zur Verfügung gestellt werden. Für GPS-Tracks wird bei OpenStreetMap das GPX-Format verwendet.

- **Lokales Wissen**
  ist für viele Informationen wie Hausnummern, Straßennamen, Verkehrsschilder und Ähnliches unbedingt erforderlich und wird meist beim Aufzeichnen der GPS-Tracks mit Papier und Bleistift notiert. Mittlerweile gibt es jedoch einige interessante Methoden um diesen Schritt zu erleichtern. So besteht beispielsweise die Möglichkeit statt dessen georeferenzierte Fotos oder Audiodaten zu verwenden.

- **OpenStreetBugs** [5]
  ist eine Karte in der Nutzer von OSM-Daten Fehler markieren können. Zum Melden eines Fehlers genügt ein Mausklick auf die betroffene Kartenstelle und ein kurzer Kommentar. Eine Anmeldung ist nicht erforderlich. OpenStreetBugs ist wohl zur Zeit die einfachste Möglichkeit an der Verbesserung von OpenStreetMap aktiv mitzuarbeiten.

[5] `http://openstreetbugs.schokokeks.org`

- **Datenspenden** [6]
  aus vielen unterschiedlichen dafür freigegebenen Quellen wurden bereits in OpenStreetMap importiert. Da die Daten meist aufwendig konvertiert werden müssen, ist dies eher eine Aufgabe für Spezialisten. Diese müssen dabei vermehrt berücksichtigen, dass OSM-Daten teilweise bereits besser als der geplante Import sind. Wichtige Datenspenden sind dabei vor allem politische Informationen wie Gemeinde- oder Kreisgrenzen, die sich mit Hilfe der anderen Datenquellen nicht erfassen lassen.

OpenStreetMap bietet einige leistungsfähige Editoren, mit denen die Daten komfortabel bearbeitet werden können. Im Folgenden wird gezeigt wie Informationen aus den verschiedenen Datenquellen mit Hilfe des JOSM-Editors [7] erfasst werden können. Eine Editiersitzung beginnt meist mit dem Laden eines Kartenausschnitts in den Editor. Für den Ausschnitt wird eine neue Datenebene angelegt, auf der man mit den Grundelementen Node und Way zeichnen kann. Einige der genannten Datenquellen können in Form von zusätzlichen Ebenen eingeblendet werden.

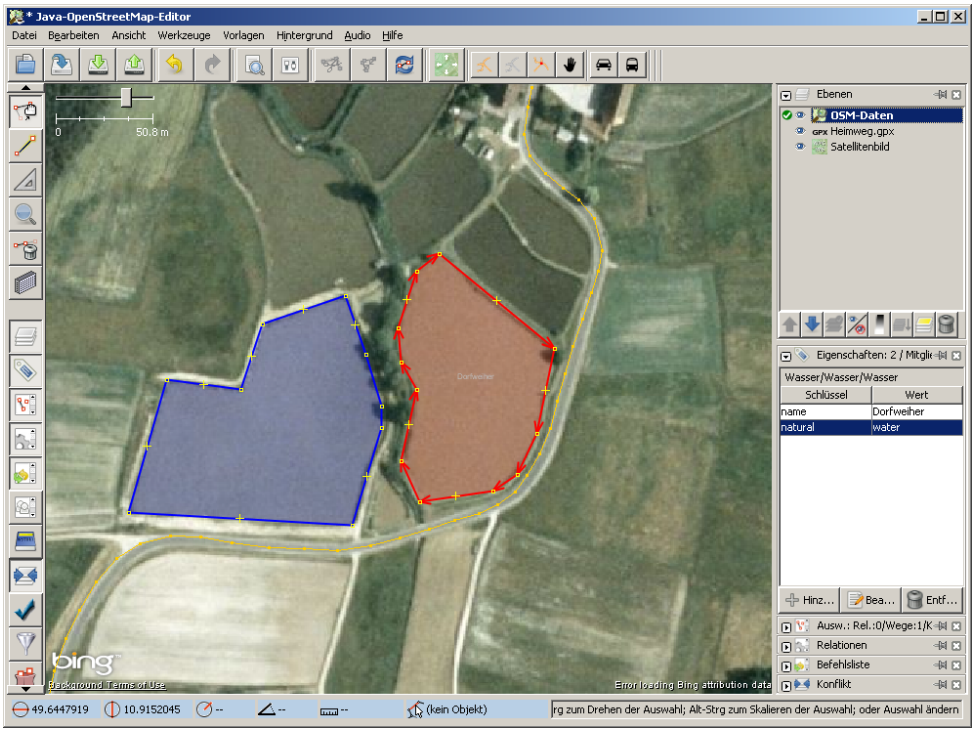


Abbildung 2.2.: JOSM - Ebenen und Tags

[6] http://wiki.openstreetmap.org/wiki/Import/Catalogue
[7] http://josm.openstreetmap.de

In Abbildung 2.2 ist ein kleiner Ausschnitt einer Karte abgebildet, in dem ein Satellitenbild und ein zuvor aufgezeichneter GPS-Track eingeblendet sind. Am rechten Bildschirmrand ist eine Auflistung der verwendeten Ebenen und die Tags des gerade ausgewählten Elements zu sehen. Es ist nicht notwendig irgendwelche Tags auswendig zu wissen. Ein Element, das in die Karte aufgenommen werden soll, wird einfach nachgezeichnet und anschließend eine passende Vorlage darauf angewendet. Ebenso einfach lassen sich Informationen aus OpenStreetBugs einblenden um die darin angegebenen Fehler zu beheben. Der Editor ermöglicht das Einbinden unterschiedlichster Karten und Satellitenbilder über sogenannte **Web Map Services** (WMS). Da OpenStreetMap einen solchen Dienst betreibt, kann natürlich auch die OSM-Karte selbst als Hintergrund ausgewählt werden. Besonders hilfreich ist die Möglichkeit, dass aufgenommene Fotos oder Audioaufnahmen automatisch einer Position auf einer GPS-Spur zugeordnet werden können. Informationen zu Straßennamen, Hausnummern und Ähnliches können dadurch komfortabel erfasst und verarbeitet werden.

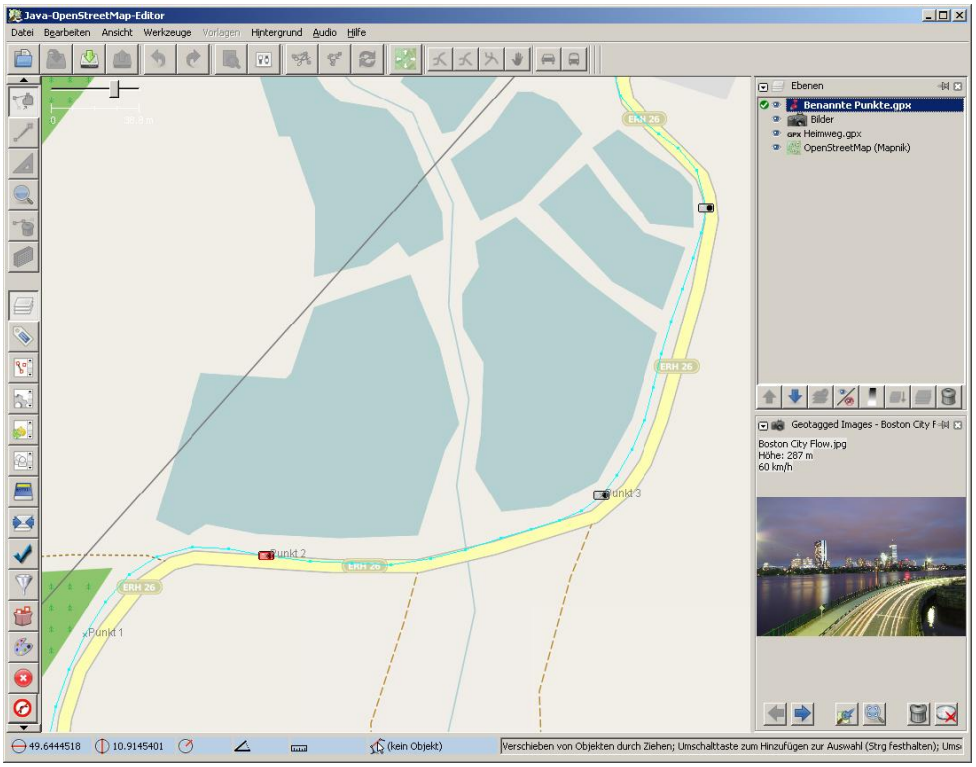


Abbildung 2.3.: JOSM - Georeferenzierte Fotos

Eine weitere Möglichkeit ist die Erstellung eines sogenannten **Walking Papers**[8]. Hier wird ein Kartenausschnitt einfach ausgedruckt, von Hand Informationen hinzugefügt gescannt und wieder hochgeladen. Anhand einer aufgedruckten Identifikation, in Form

[8] `http://walking-papers.org`

eines Barcodes, kann ermittelt werden um welchen Kartenbereich es sich dabei handelt. Ähnlich wie bei OpenStreetBugs, ist hierzu keine Anmeldung erforderlich. Auch für das Erstellen von Relationen gibt es Hilfsmittel, jedoch sind diese noch nicht sehr ausgereift. Für einige Relationen gibt es zwar Vorlagen, die zumindest das Setzen der benötigten Tags übernehmen, die Role der einzelnen Member muss jedoch meist von Hand eingetragen werden. Multipolygone können hingegen sehr einfach, durch Markierung der beteiligten Elemente und den Aufruf von *Multipolygon erstellen* erzeugt werden.

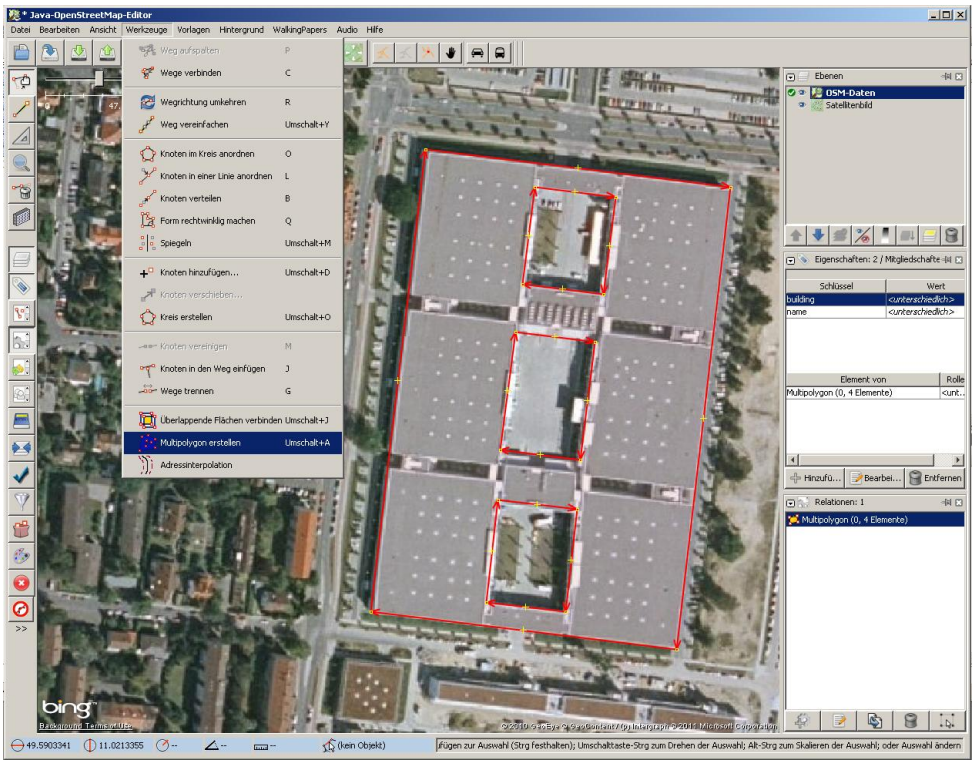


Abbildung 2.4.: JOSM - Erstellung eines Multipolygons

Noch eine kurze Anmerkung zu Relationen, da dies in der Literatur nicht deutlich erwähnt wird: Wenn ein Node oder Way Teil einer Relation ist, so wird dies im Eigenschaftsfenster angezeigt (Abbildung 2.4 rechts in der Mitte). Ist ein solches Element verändert worden sollte die Relation überprüft werden, da sich sonst Fehler einschleichen können.

2.3. OpenStreetMap - API

Alle OpenStreetMap-Daten werden in einer zentralen PostgreSQL-Datenbank gespeichert. Der Zugriff auf die Daten erfolgt aber nicht direkt, sondern über eine auf dem **Hypertext Transfer Protocol** (HTTP) basierende Schnittstelle. Informationen werden hierbei in einem Format übertragen, das auf der **Extensible Markup Language** (XML) basiert. Für <typ> in Tabelle 2.4 können die Werte `node`, `way`, `relation` und **changeset** eingesetzt werden, die bis auf Changeset bereits vorgestellt wurden. Mit einem Changeset können mehrere Änderungen an den OSM-Daten zusammengefasst und in einer einzigen Anfrage übertragen werden. Es ist ebenfalls hilfreich, wenn zusammenhängende Änderungen wieder rückgängig gemacht werden sollen.

Anfrage	HTTP-Methode und URI	Anfrage	Antwort
Serverfähigkeiten	GET /api/0.6/capabilities	-	XML
Elelement anlegen	PUT /api/0.6/<typ>/create	XML	id
Element abfragen	GET /api/0.6/<typ>/<id>	-	XML
Element löschen	DELETE /api/0.6/<typ>/<id>	XML	-
Element ändern	PUT /api/0.6/<typ>/<id>	XML	version
Historie	GET /api/0.6/<typ>/<id>/history	-	XML
Ältere Version	GET /api/0.6/<typ>/<id>/<version>	-	XML
Mehrere Elemente eines Typs	GET /api/0.6/<typ>s/<typ>s= <id>[,<id>, ...]	-	XML
Referenzierte Elemente	GET /api/0.6/<typ>/<id>/<typ>s	-	XML
Element komplett	GET /api/0.6/<typ>/<id>/full	-	XML
Elemente mit Tag finden	GET /api/0.6/<typ>s/search? type=<key>&value=<value>	-	XML
Mehrere Elemente ändern	POST /api/0.6/changeset/ <id>/upload	XML	XML
Kartenbereich abfragen	GET /api/0.6/map? bbox=<left>,<bottom>,<right>,<top>	-	XML

Tabelle 2.4.: OpenStreetMap-API für Datenelemente

Im Folgenden sollen nur die für die vorliegende Arbeit relevanten Teile der API betrachtet werden. Speziell interessiert das XML-Format für die Grundelemente Node, Way und Relation. Als HTTP-Methode wird hier lediglich die GET-Methode verwendet, die einem Aufruf des **Uniform Resource Identifiers** (URI) in der Adressleiste eines Browsers entspricht.

Jedes Element in OpenStreetMap besitzt eine 64 Bit breite als **id** bezeichnete weltweit eindeutige Kennung, die nach dem Löschen nicht neu vergeben wird. Ebenfalls enthalten sind der Zeitpunkt der letzten Änderung, Benutzernummer und -kennung des letzten Bearbeiters, die aktuelle Version und das Changeset der letzten Änderung. Optional besitzt jedes Element in OpenStreetMap eine beliebige Menge von Tags. Listing 2.1 enthält das XML-Format für eine Node. Eine Node besitzt eine geometrische Länge und Breite die mit **lat** (Latitude) und **lon** (Longitude) angegeben werden. Sie werden als Dezimalzahlen mit sieben Nachkommastellen gespeichert, was einer Genauigkeit von etwa 1 cm entspricht.

```
1  <osm version="0.6" generator="OpenStreetMap server">
2      <node id="919683840" lat="49.6469557" lon="10.9162722" version="5" changeset="8978020" user
           ="bobbery" uid="428563" visible="true" timestamp="2011-08-10T16:21:06Z">
3          <tag k="name" v="Gasthaus Jägerheim"/>
4          <tag k="addr:postcode" v="91093"/>
5          <tag k="addr:housenumber" v="5"/>
6          <tag k="addr:city" v="Röhrach"/>
7          <tag k="addr:street" v="Dechsendorfer Straße"/>
8          <tag k="amenity" v="restaurant"/>
9          <tag k="addr:country" v="DE"/>
10         <tag k="phone" v="09135/8716"/>
11     </node>
12 </osm>
```

Listing 2.1: XML-Darstellung einer Node

Listing 2.2 repräsentiert einen Way. Er besitzt eine geordnete Liste mit Referenzen auf Nodes, die dessen Form beschreiben.

```
1  <osm version="0.6" generator="OpenStreetMap server">
2      <way id="38766465" visible="true" timestamp="2010-09-20T18:13:33Z" version="2" changeset="
           5831242" user="kanu_guenni" uid="73317">
```

```
3         <nd ref="410404152"/>
4         <nd ref="919690360"/>
5         <nd ref="26152914"/>
6         <tag k="highway" v="residential"/>
7         <tag k="name" v="Bruckäckerweg"/>
8     </way>
9 </osm>
```

Listing 2.2: XML-Darstellung eines Ways

Listing 2.3 beschreibt eine Relation, die aus Mitgliedern verschiedenen Typs und Funktion bestehen kann. Im Beispiel ist ein Multipolygon (zusammengesetzte Fläche) beschrieben, das aus mehreren Ways besteht. Die Ways mit der Role `outer` bestimmen hierbei die Außen- und die mit der Role `inner` die Innenflächen.

```
1 <osm version="0.6" generator="OpenStreetMap server">
2     <relation id="195644" visible="true" timestamp="2010-06-04T15:31:09Z" version="4" changeset
        ="4902031" user="Nightdive" uid="95293">
3         <member type="way" ref="26276526" role="inner"/>
4         <member type="way" ref="33084925" role="inner"/>
5         <member type="way" ref="33084930" role="inner"/>
6         <member type="way" ref="3678599" role="outer"/>
7         <member type="way" ref="26276513" role="inner"/>
8         <member type="way" ref="26276525" role="inner"/>
9         <tag k="type" v="multipolygon"/>
10    </relation>
11 </osm>
```

Listing 2.3: XML-Darstellung einer Relation

Einer der wichtigsten API-Aufrufe ist die Abfrage aller Elemente in einem geografischen Bereich, die auch als OSM-Dump bezeichnet wird.

Die Antwort des Servers auf diese Anfrage hat folgende Form:

- **Alle Nodes,**
  die im gewählten Bereich enthalten oder zu einem enthaltenen Way gehören
- **Alle Ways,**
  von denen sich ein Teil im gewählten Bereich befindet

- **Alle Relationen,**
  die mindestens einen der enthaltenen Nodes oder Ways als Member haben

Diese Form hat auch das sogenannte Planet-File, eine XML-Datei mit dem kompletten Inhalt der OSM-Datenbank, welches einmal wöchentlich erstellt wird. Einige Anbieter wie z. B. die Geofabrik [9] erzeugen aus diesen Daten Auszüge für einzelne Länder, die als Basis für den in dieser Arbeit entwickelten Konverter dienen.

2.4. Lizenz

Die Daten in der OSM-Datenbank befinden sich unter der *Creative Commons Attributation-Share-Alike*-Lizenz, [10] was so viel bedeutet wie *Namensnennung (des Rechteinhabers) und Weitergabe unter gleichen Bedingungen*. Der Rechteinhaber ist hierbei nicht die OpenStreetMap-Foundation, sondern alle Benutzer die Daten zum Projekt beigetragen haben. Entsprechend sind abgeleitete Werke mit der Kennzeichnung

Map Data © OpenStreetMap contributors, CC-BY-SA

zu versehen. Die Daten des OpenStreetMap-Projekts dürfen beliebig verwendet, weiterverarbeitet und (auch gegen Bezahlung) veröffentlicht werden. Im Falle einer Veröffentlichung müssen den Nutzern jedoch wieder die selben Rechte zugesprochen werden. Leider hat sich herausgestellt, dass CC-BY-SA die Daten nicht ausreichend schützt, da in vielen Ländern Fakten wie Telefonnummern oder auch Geodaten nicht schutzwürdig sind. Sammlungen von Fakten sind dies jedoch schon, weshalb derzeit ein Lizenzwechsel auf die *Open Database Licence* (ODbL) [11] angestrebt wird. Die ODbL ist etwas freizügiger für abgeleitete Werke und erlaubt beliebige Lizenzierungen. Abgeleitete Datenbanken müssen jedoch, auch wenn sie nur als Vorstufe für ein publiziertes Werk verwendet werden, veröffentlicht und freigegeben werden. Es kann jedoch, statt der ganzen Datenbank, auch nur der Algorithmus zum Erzeugen der Daten veröffentlicht werden.

[9] http://www.geofabrik.de
[10] http://creativecommons.org/licenses/by-sa/2.0/
[11] http://opendatacommons.org/licenses/odbl/

2.5. Anwendungen

Die durch das OpenStreetMap-Projekt gesammelten Daten können für viele unterschiedliche Anwendungen verwendet werden, von denen hier nur einige wenige beschrieben werden. Online-Karten sind das Hauptanwendungsgebiet der OpenStreetMap-Daten und die Erstellung thematisierter Karten eine der größten Stärken. Es gibt bereits viele Spezialkarten, die auf ein gewisses Anwendungsgebiet zugeschnitten sind, wie zum Beispiel:

- **OpenCycleMap**
  zeigt Fahrradrouten - große Straßen rücken in den Hintergrund
  `http://www.opencyclemap.org`
- **ÖPNV-Karte**
  zeigt Verbindungen durch öffentliche Verkehrsmittel und Haltestellen
  `http://www.öpnvkarte.de`
- **Wheelmap**
  zeigt behindertengerechte Einrichtungen und Wege
  `http://wheelmap.org`

Ein Beispiel für die kommerzielle Nutzung von OSM-Daten lässt sich auf der Seite `http://www.digitalradio.de` finden, auf der eine Karte Sendestationen und deren Empfangsgebiete für digitales Radio darstellt. Der Benutzer kann seinen Wohnort eingeben um zu erfahren, ob dieser Dienst in seiner Nähe verfügbar ist (Abbildung 2.5). Ein weiteres interessantes Anwendungsgebiet ist die Nutzung von OpenStreetMap zur Hilfe bei Katastrophen. So wurden die Hilfskräfte, nach einem starken Erdbeben in Haiti, durch eine OSM-Karte koordiniert. Diese wurde anhand von Satellitenbildern und Informationen der Hilfsorganisationen, in kürzester Zeit von Mappern auf der ganzen Welt, erstellt. Enthalten waren beispielsweise Informationen zu zerstörten Gebäuden und Straßen, Hilfscamps oder Wasserentnahmestellen [12].

Wer selbst eine thematische Karte erstellen möchte, kann dabei auf vorhandene **Renderer** wie Mapnik, Osmarender oder Kosmos zurückgreifen und diese nach eigenen Wünschen konfigurieren. Natürlich können diese, oder auch bereits existierende Karten, in eine Homepage integriert oder ausgedruckt werden. Daneben lassen sich OSM-Daten auch bereits zur Routenberechnung nutzen. Neben einigen online Routingdiens-

[12] `http://wiki.openstreetmap.org/wiki/WikiProject_Haiti`

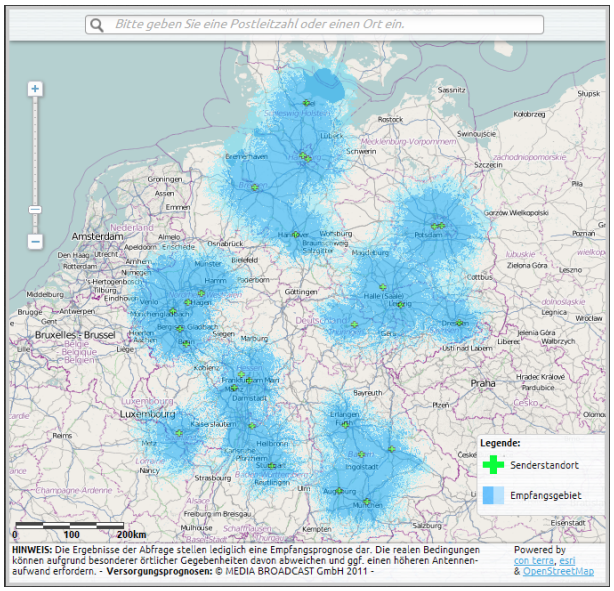


Abbildung 2.5.: Empfangsbereiche für digitales Radio

ten wie dem OpenRouteService [13] der Universität Heidelberg oder YourNavigation [14] gibt es auch eine große Anzahl an mobilen Lösungen. Viele Applikationen wie OsmAnd, Skobbler oder Roadee nutzen für das Routing und die Zieleingabe einen Online-Dienst und benötigen hierzu eine Internetverbindung. Cloudmade bietet beispielsweise einen solchen Dienst über eine einfache API[15] an. Für die vorliegende Arbeit sind jedoch Navigationslösungen von Interesse, die ohne eine Internetverbindung auskommen:

- **mkgmap[16]**
  ist ein freier Konverter von OSM-Daten für Navigationsgeräte des Herstellers Garmin. Dieser wurde jedoch ohne Zutun der Firma von einigen Enthusiasten, durch Reverse Engineering des zugrundeliegenden Formates, realisiert. Da das Format in einigen Details noch nicht verstanden ist, gibt es noch einige Probleme. So können auf direktem Wege keine Karten mit Adresssuche generiert werden und das Routing ist noch experimentell. Etwas bessere Ergebnisse verspricht eine kommerzielle Lösung [17], die aber die Benutzung der generierten Daten einschränkt und natürlich etwas kostet.

[13] http://openrouteservice.org
[14] http://www.yournavigation.org
[15] http://developers.cloudmade.com/projects/show/routing-http-api
[16] www.mkgmap.org.uk
[17] http://www.cgpsmapper.com

- **MoNav**[18]
  basiert auf einer Diplomarbeit von Sebastian Vetter und besticht durch seinen einfachen Aufbau und einer hohen Geschwindigkeit bei Kartendarstellung und Routenberechnung. MoNav ist nur von der Qt-Bibliothek abhängig und kann daher auf viele Plattformen portiert werden [19]. Das Programm ist sehr modular aufgebaut und viele Teile können in Form von Plugins ausgetauscht werden. So wurden in der ersten Version für die Kartendarstellung vorberechnete Bitmaps verwendet, die mittlerweile durch einen eingebauten Renderer ersetzt wurden. Die berechnete Route wird einfach darüber gezeichnet. Ein großer Teil der Funktionalität steckt in einem Vorverarbeitungsschritt. Ein Preprozessor generiert aus den OpenStreetMap-Daten Datenstrukturen, die für das Routing und die Zieleingabe geeignet sind. (Sogar das Kantengewicht mit dem der Routing-Algorithmus arbeitet wird bereits hier, für alle möglichen Fahrzeugtypen, ermittelt.) Leider hat MoNav auch einige Nachteile. So werden beispielsweise keine Straßennamen und Points Of Interest angezeigt. Die Zieleingabe beschränkt sich auf eine Angabe von Ort und Straße. Dazu werden den Orten Straßen zugeordnet, die sich in deren Nähe befinden. Der Radius hierfür variiert hierbei mit der Größe des Ortes (Stadt hat größeren Radius als Ortschaft). Dadurch können Straßen beispielsweise in Orten vorkommen, zu denen sie eigentlich nicht gehören. Gesprochene Fahranweisungen gibt es nicht und die benötigten Daten sind sehr groß (ca. 6 GB für Deutschland).

- **Gosmore**[20]
  ist ebenfalls eine sehr interessante Lösung. Die Kartenanzeige ist sehr reich an Details und es gibt sogar eine einfache 3D-Darstellung. Zoomt man etwas aus der Karte heraus, so häufen sich aber Darstellungsfehler. Straßen erscheinen dann als unterbrochen oder parallel verlaufende Fahrbahnen überschneiden sich. Die Adresssuche funktioniert nicht hierarchisch nach Ort und Straße, sondern liefert eine Liste mit Entsprechungen für eine eingegebene Zeichenkette mit der Entfernung zur aktuellen Position. Bei der Eingabe können auch Tags wie `bank`, `restaurant` oder `shop` verwendet werden um nach POIs zu suchen. Start- und Endpunkt einer Route müssen jedoch von Hand in der Karte markiert werden. Wird bei der Navigation die Route verlassen, so wird diese nicht neu berechnet. Einfache Fahranweisungen werden gegeben. Allgemein benötigt die Anwendung sehr viel Rechenleistung und ist sehr unstabil. Die Bedienung ist leider nicht sehr

[18] `http://monav.openstreetmap.de/`
[19] `http://doc.qt.nokia.com/latest/supported-platforms.html`
[20] `http://wiki.openstreetmap.org/wiki/Gosmore`

benutzerfreundlich und das Design der Software nicht besonders modular. Gosmore verwendet ein eigenes Dateiformat [21], das hier nicht näher beschrieben wird. Ein Bereich von der Größe Bayerns benötigt etwa 500 MB. Viel größere Karten sollten, aus Performance-Gründen, auch nicht verwendet werden.

- **Navit**[22]
  ist wohl derzeit die ausgereifteste Navigationslösung auf OSM-Basis. Hier wird wieder ein eigenes Dateiformat verwendet, das die Welt in Tiles (Kacheln) aufteilt. Auf der höchsten Zoomstufe gibt es 4 Kacheln, die jeweils wieder in 4 Kacheln zerlegt werden. Das ganze passiert bis zu 14 mal. Ein OSM-Objekt (beispielsweise ein Way) wird hierbei anhand seiner Koordinaten einem Tile zugeordnet, in das es gerade noch hineinpasst. Durch dieses Vorgehen können die Daten für das Rendern der Karte gezielt und schnell geladen werden ohne dabei viel Speicher zu benötigen. Das Dateiformat ist dank ZIP-Komprimierung sehr kompakt und benötigt für Deutschland etwa 300 MB Speicherplatz. Navit besitzt eine 2D- und 3D- Kartendarstellung, die zwar etwas pixelig ist, sich aber dennoch sehen lassen kann. Über **Text To Speech** (TTS) können sehr genaue Fahranweisungen gegeben werden. Hierzu wird jedoch ein externes Programm verwendet. Die Adresssuche funktioniert wie bei MoNav und verursacht daher ähnliche Probleme, wobei hier auch POIs und Hausnummern (soweit in OSM vorhanden) gewählt werden können. Durch den modularen Aufbau und die Unterstützung verschiedenster GUI-Toolkits lässt sich Navit auf fast alle gängigen Plattformen portieren. Die Android-Portierung wurde bereits über 100 000-mal heruntergeladen und von vielen Benutzern als sehr gut bewertet. Die Bedienung und die Adresseingabe wird jedoch noch oft bemängelt [23].

Diese Liste erhebt keinen Anspruch auf Vollständigkeit und stellt lediglich einige interessante Lösungsansätze vor. Mehr Informationen zum Thema Offline-Routing gibt es im OSM-Wiki [24].

[21] http://wiki.openstreetmap.org/wiki/Gosmore/Internals
[22] http://www.navit-project.org/
[23] https://market.android.com/details?id=org.navitproject.navit&feature=search_result
[24] http://wiki.openstreetmap.org/wiki/Routing/offline_routers

2.6. Datenqualität

Mit Schlagzeilen wie „OpenStreetMap zieht an kommerziellem Anbieter vorbei“ [25] sorgte die Studie [Zielstra, 2010] im Dezember 2010 für aufsehen. In ihr hatten die Autoren die Daten aus dem OSM-Projekt mit denen des kommerziellen Kartenanbieters TeleAtlas verglichen. Dabei wurde festgestellt, dass die OSM-Daten in großen deutschen Städten bereits detaillierter als die des kommerziellen Anbieters sind. Zur Qualität der Daten zählt jedoch nich nur die Menge der vorhandenen Daten, sondern auch deren Genauigkeit. So erreicht TeleAtlas, mit moderner Technik, eine Genauigkeit von etwa einem Meter. Die, mit einfachen Navigationsgeräten ermittelten, OSM-Daten hingegen, im besten Fall, nur etwa 5 Meter. Abbildung 2.6 zeigt, wie viele Kartenelemente in OSM eine Abweichung von weniger als 10 m zu den Daten von TeleAtlas aufweisen. Während der Vergleich in großen Städten sehr positiv ausfällt, gibt es in kleineren Städten aber bereits größere Abweichungen. Allgemein ist zu bemerken, dass die Qualität der OSM-Daten stark von der Anzahl der Mapper in einem Gebiet abhängt.

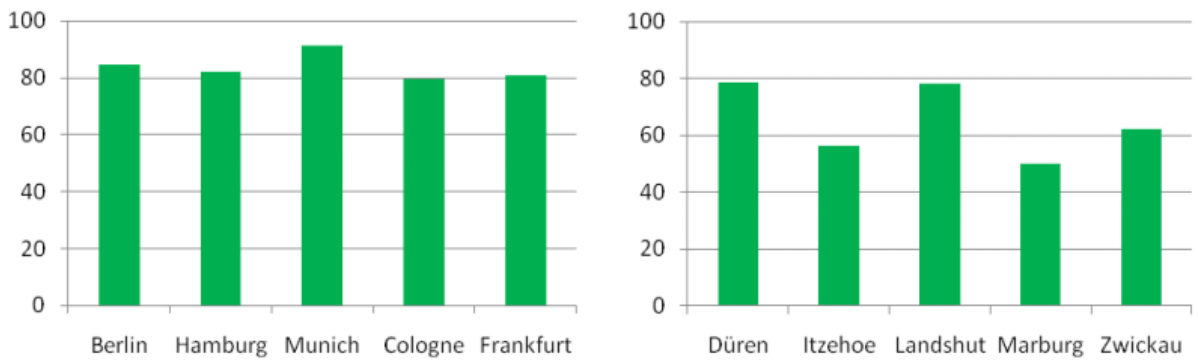


Abbildung 2.6.: Abweichungen von der Position kleiner 10 m

[25] www.focus.de, www.handelsblatt.de, www.n24.de

Ein anderer Faktor der verglichen wurde, ist die Länge des vorhandenen Straßennetzes. Während ein Vergleich der alle Fuß-, Rad- und Feldwege mit einschließt mehr zugunsten von OpenStreetMap war, hat TeleAtlas bei reinen Straßendaten aber immer noch die Nase vorn. Wie Abbildung 2.7 illustriert, entwickelt sich das OSM-Straßennetz jedoch sehr schnell und wird TeleAtlas in wenigen Jahren einholen.

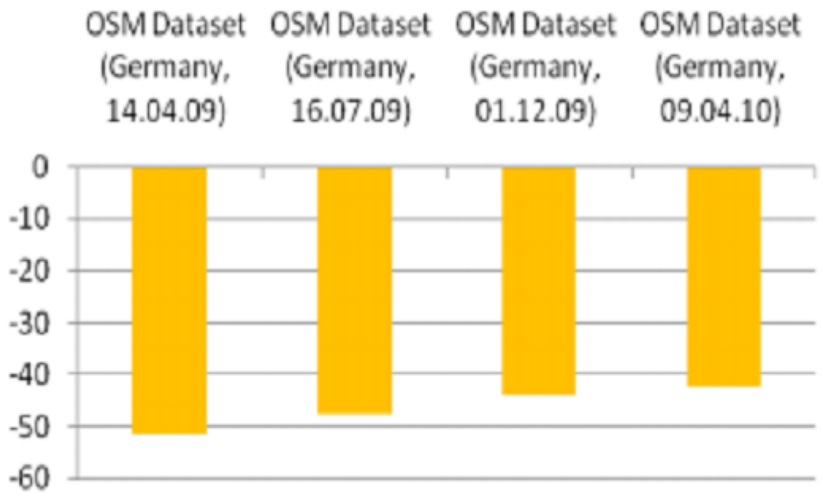


Abbildung 2.7.: Länge des OSM-Straßennetzes im Vergleich zu TeleAtlas

3. Elektrobit Automotive

Elektrobit (EB) ist ein führender Entwickler von innovativen Embedded-Lösungen für die Fahrzeug und Wireless-Industrie. Das Geschäftsgebiet Automotive Software hat sich international als einer der wichtigsten Anbieter von Embedded-Software-Lösungen für die Automobilindustrie etabliert. EB bietet eine breite Palette an ausgereiften Softwarelösungen in den Bereichen Embedded Software für Steuergeräte, Fahrerassistenzsysteme, Infotainment- und Navigationslösungen. Die Navigationslösung von EB heißt **EB street director**.

3.1. EB street director

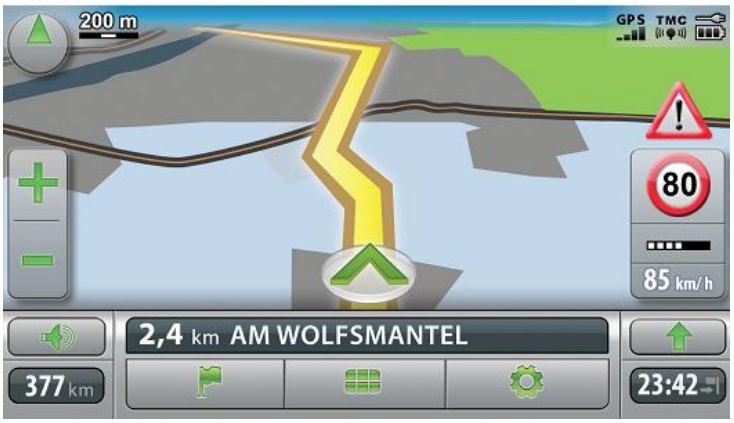


Abbildung 3.1.: EB street director Application

Mit dem **EB street director** bietet Elektobit seinen Kunden eine ausgereifte Navigationslösung an. Die Software wird dabei nach dem White-Label-Prinzip vertrieben. Das bedeutet, dass Kunden das Produkt unter einem eigenen Namen anbieten können. Beispiele für Anwendungen des **EB street director** sind portable Navigationsgeräte von Falk und Medion oder die OnBoard-Navigation vieler bekannter Automobilhersteller. Die Fähigkeiten des **EB street director** gehen dabei weit über die Möglichkeiten der, bereits vorhandenen, Navigationslösungen auf OpenStreetMap-Basis hinaus.

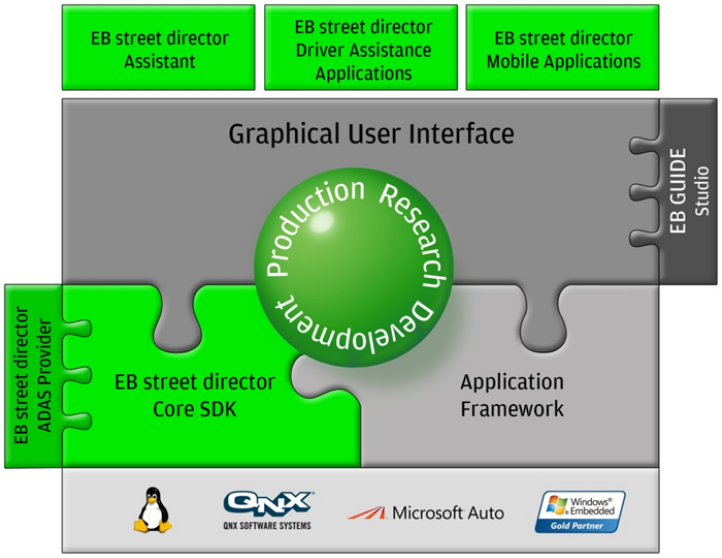


Abbildung 3.2.: Elektrobit StreetDirector

Der **EB street director** lässt sich in vielen Details an die Wünsche der Kunden anpassen. Er ist entweder als fertige Applikation mit Benutzerschnittstelle oder als **Software Development Kit** (SDK) erhältlich. Die Verwendung als SDK bietet hierbei maximale Flexibilität, wogegen die Applikation eine etablierte Standardlösung bereitstellt. Die Kartendarstellung sowie die Menüführung können dabei, ohne Programmieraufwand, an die Anforderungen des Kunden angepasst werden. Für das Entwerfen von zuverlässigen grafischen Oberflächen und Sprachdialogen bietet Elektrobit hierzu mit **EB guide** eine hervorragende Toolunterstützung an. Die Menüführung kann dabei grafisch, in Form von UML-Zustandsdiagrammen, modelliert werden (Abbildung 3.3).

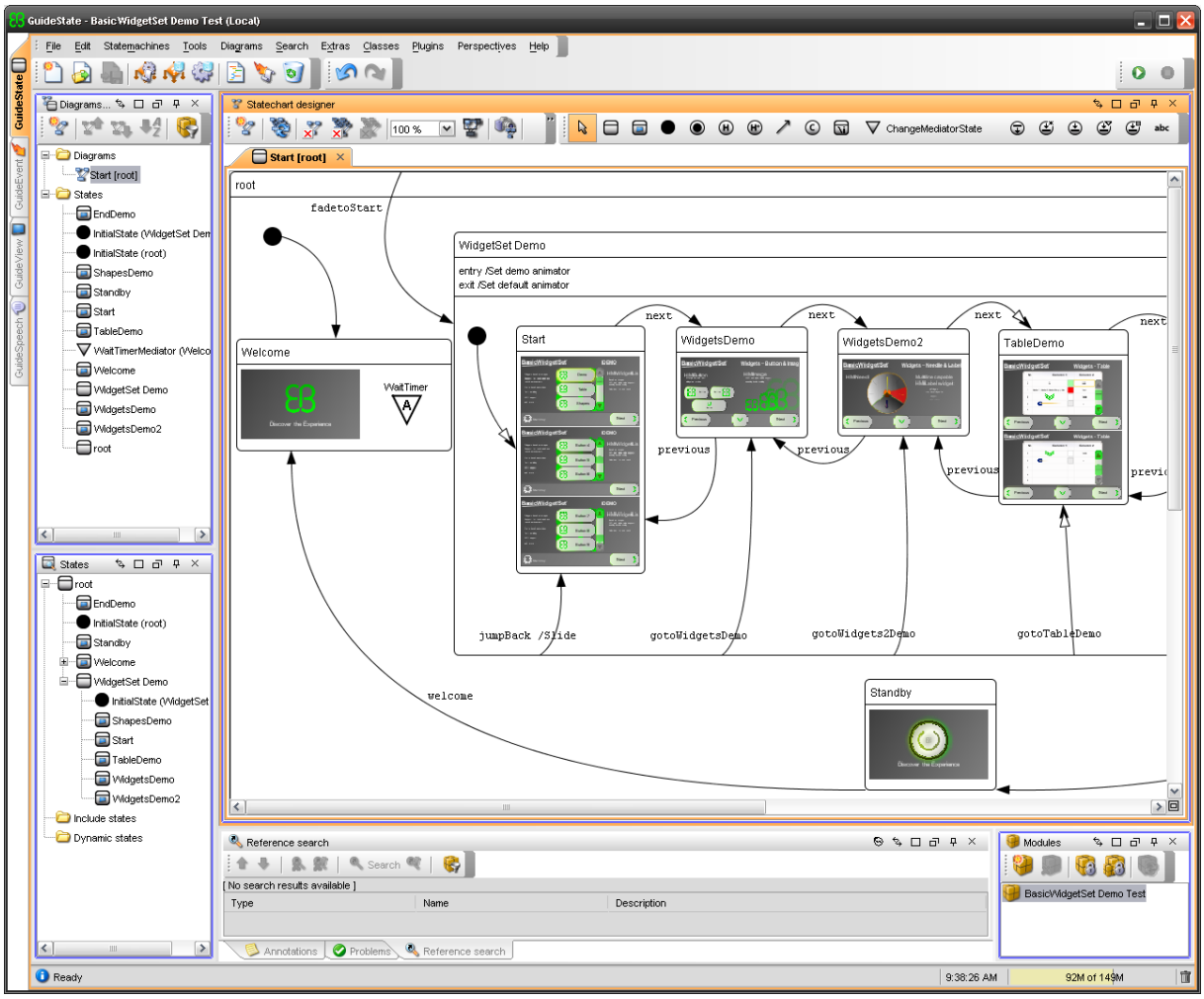


Abbildung 3.3.: EB Guide

Über eine Reihe von Schnittstellen lässt sich der **EB street director** um kundenspezifische Funktionen erweitern. Über eine ADAS-Schnittstelle[26] können die Navigationsdaten beispielsweise als elektronischer Horizont verwenden werden, um Fahrerassistenzfunktionen wie eine Kurvenwarnung oder Kurvenlicht zu realisieren. Zusätzlich lassen sich in die Umgebung Applikationen wie MP3-Player, Adressbuch oder Video-Player einbetten. Die Kartendarstellung des **EB street director** ist in der Lage Modelle von Städten oder Höheninformationen (DTM) dreidimensional darzustellen und bietet eine realistische Kreuzungsvorschau. Mit Windows Embedded, Microsoft Auto, QNX und Linux wird ein Großteil der aktuellen Softwareplattformen unterstützt.

Abbildung 3.4.: Kartendarstellungen

Im Zusammenhang mit dem **EB street director** gibt es noch einige nützliche Werkzeuge, die hier kurz vorgestellt werden. Mit dem **EB street director Assistant** können die Hersteller von Navigationsgeräten ihren Kunden eine benutzerfreundliche Konfiguration ihres Gerätes und die Installation von Kartenupdates ermöglichen. Mit Hilfe des **DesktopNavigators** können auf einfache Weise die Funktionen des SDKs erkundet werden. Hierfür steht unter anderem eine einfache Skript-Schnittstelle zur Verfügung. Mit dem **MapConfigurator** ist eine einfache Anpassung der Kartendarstellung möglich.

[26] Advanced driver assistance systems

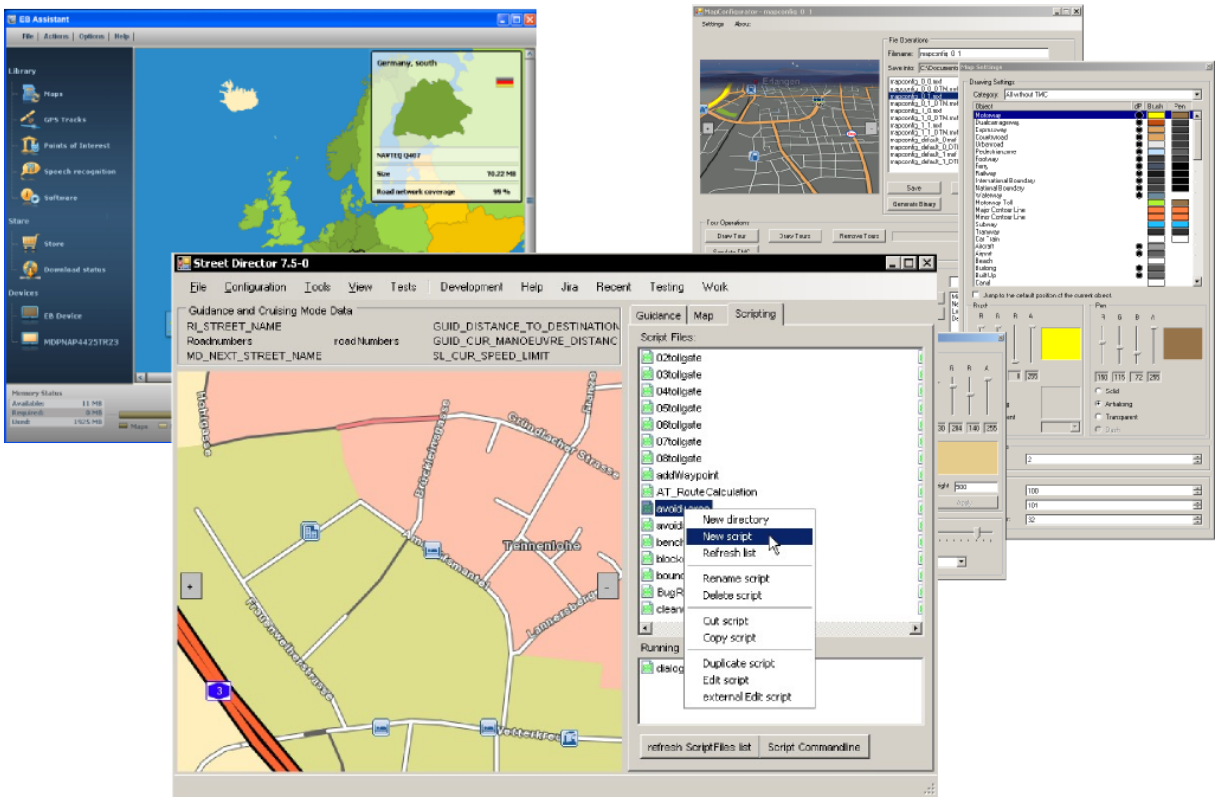


Abbildung 3.5.: Tools für den **EB street director**

3.2. Kartenübersetzungsprozess

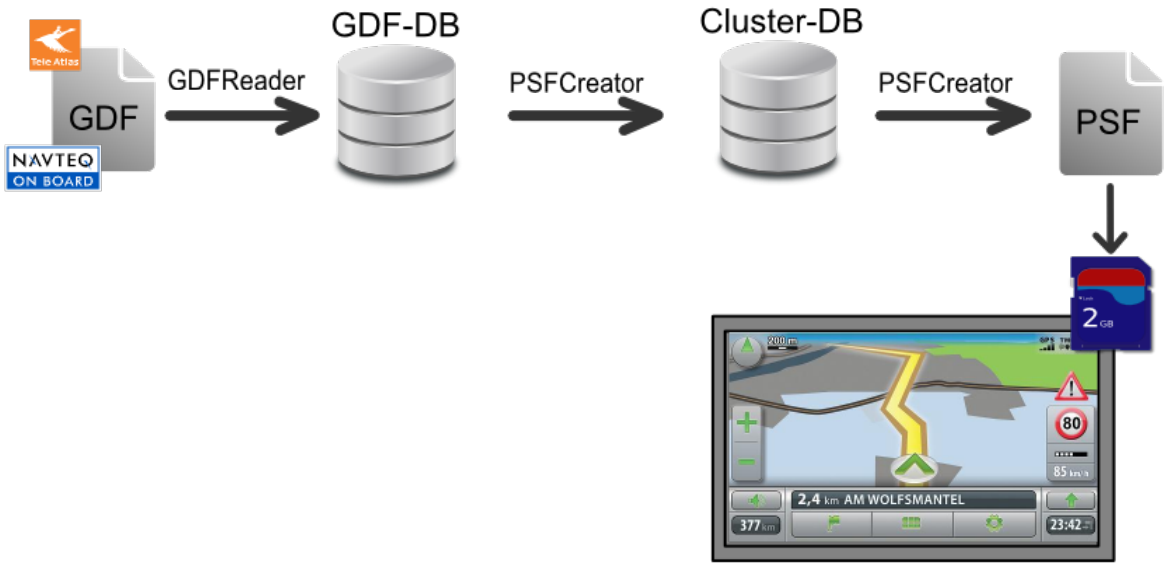


Abbildung 3.6.: Kartenübersetzungsprozess bei EB

Elektrobit verwendet für den **EB street director** ein eigenes Dateiformat, das als **Physical Storage Format** (PSF) bezeichnet wird. In regelmäßigen Abständen werden von kommerzielen Anbietern für Navigationsdaten neue Karten als **Geographic Data Files** (GDF) zur Verfügung gestellt. GDF ist ein Textformat für vektorisierte Kartendaten, das sich bereits seit vielen Jahren im Navigationsbereich etabliert hat. In einem eigenen Kartenübersetzungsprozess werden diese Daten zuerst in eine GDF-Datenbank eingelesen, die das GDF-Format sinngemäß in einer Datenbankstruktur nachbildet. Als Nächstes werden diese Daten in eine sogenannte Cluster-Datenbank eingelesen. Dabei werden die Daten neu strukturiert und Informationen für Routenberechnung, Kartendarstellung oder Zieleingabe daraus extrahiert. Die Daten dieser logischen Abschnitte werden dann in sogenannte Cluster organisiert. Ein Cluster kann dabei als ein kleiner Kartenausschnitt betrachtet werden, in dem die jeweiligen Daten enthalten sind. Dies wird benötigt um später schnell auf die benötigten Daten zugreifen zu können und bildet die Grundlage für das PSF-Format, welches anschließend daraus erzeugt wird.

3.3. Physical Storage Format (PSF)

Abbildung 3.7 zeigt den grundsätzlichen Aufbau einer PSF-Datei. Diese besteht aus mehreren logischen Blöcken. Die Daten sind dabei in Datenstrukturen organisiert, die einen schnellen Zugriff ermöglichen. So können beispielsweise Daten für die Kartendarstellung anhand der Zoomstufe und dem darzustellenden Kartenbereich abgefragt werden. Dank ZIP oder LZMA-Komprimierung und Datenreduktion auf Basis des Douglas-Peucker Algorithmus ist das PSF-Format sehr kompakt. Hier werden nur einige der wichtigsten Bestandteile einer PSF-Datei beschrieben:

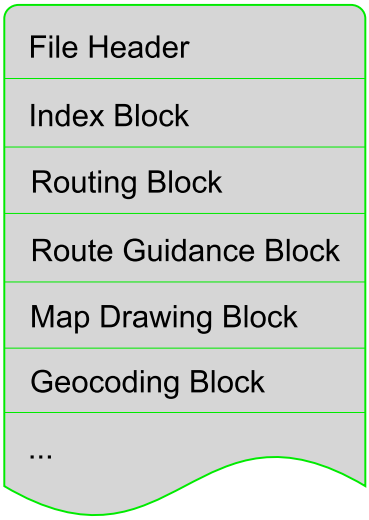


Abbildung 3.7.: Aufbau des PSF-Formats

- **File Header**
  enthält Meta-Informationen zum Aufbau der Datei. Ebenso befinden sich hier globale Eigenschaften wie der geografische Bereich der durch die Datei abgedeckt wird, um welches Land es sich handelt oder von welchem Hersteller die Daten stammen.

- **Index Block**
  Auf die Inhalte der einzelnen Blöcke kann, mit Hilfe von geeigneten Datenstrukturen (B*- und R-Tree), effizient zugegriffen werden. Der Index Block enthält im Wesentlichen die Startadressen der jeweils verwendeten Strukturen, die sich innerhalb der unterschiedlichen Blöcke befinden.

- **Routing Block**
  Enthält Informationen, die zur Berechnung einer Route benötigt werden. Um eine schnelle Routenberechnung zu ermöglichen, sind die Daten in diesem Block auf

das Nötigste reduziert. Eine Straße wird hierfür nur als eine direkte Verbindung zwischen zwei Punkten betrachtet.

- **Route Guidance Block**
  Dieser Block enthält etwas genauere Informationen über eine berechnete Route und enthält beispielsweise den Verlauf der, mit den Informationen aus dem Routing Block, ermittelten Straßen. Diese Daten können für Fahranweisungen oder eine sehr einfache Kartendarstellung verwendet werden.

- **Map Drawing Block**
  enthält Daten, in Form von Punkten, Linien und Polygonen, die für das Zeichnen einer Karte benötigt werden. Für eine effiziente Darstellung können die Daten anhand des darzustellenden Bereichs und der verwendeten Zoomstufe abgefragt werden. Ebenso können Informationen für eine 3D-Darstellung enthalten sein.

- **Geocoding Block**
  enthält Informationen, die für das Berechnen einer Position anhand einer Adresse benötigt werden. Hierzu wird ein abstraktes Datenmodell verwendet, mit dem die Verwaltungsstruktur verschiedener Länder abgebildet werden kann.

4. Geographic Data Files

Geographic Data File (GDF) ist ein in der Norm **EN ISO 14825** beschriebenes, nicht binäres, Austauschformat für vektorisierte Kartendaten, dessen erste Version bereits Ende der achziger Jahre verabschiedet wurde. Der Standard beschreibt ein Referenzmodell für alle Organisationen, die sich mit der Erfassung oder Verarbeitung von digitalen Kartendaten beschäftigen und erleichtert so den Datenaustausch. Neben einer Beschreibung des zugrundeliegenden Datenmodells enthält dieser auch einen umfangreichen Katalog an möglichen Kartenelementen und Attributen, welche auch um benutzerdefinierte Elemente erweitert werden können. An vielen Stellen werden Hinweise gegeben, wie bestimmte Sachverhalte im Datenmodell nachzubilden sind.

In diesem Kapitel werden nur die grundlegenden Ideen des Standards beschrieben, da eine zu genaue Betrachtung aller Details für das Verständnis der vorliegenden Arbeit eher hinderlich wäre. Mit Hilfe der, hier beschriebenen, Elemente lassen sich nämlich durchaus sehr komplexe Strukturen bilden. Sind genauere Informationen nötig, so werden diese an der betreffenden Stelle beschrieben.

4.1. Datenmodell

Abbildung 4.1 zeigt eine sehr vereinfachte Darstellung des zugrundeliegenden Datenmodells[27]. Dabei werden nur die Elemente beschrieben, welche die eigentlichen Daten des GDF-Formats enthalten. Metadaten, die beispielsweise die verwendeten Attribute oder Kartenelemente beschreiben, wurden dabei weggelassen. Das wichtigste Konzept im GDF-Format sind die sogenannten Layer, die die Daten auf unterschiedlichen Abstraktionsebenen betrachten.

Im **Layer 0** werden die grafischen Primitive beziehungsweise die Geometrie der Daten beschrieben. Diese bestehen aus Punkten (**Nodes**), Linien (**Edges**) und Flächen (**Faces**). Punkte haben dabei eine Koordinate (**XYZ**). Linien beginnen und enden an

[27] [NAVTEQ, 2010]

einem Punkt und besitzen eine Liste mit Koordinaten, die deren Verlauf beschreiben. Flächen bestehen wiederum aus Linien, die den Verlauf von Innen- und Außenflächen beschreiben. Als Koordinatensystem wird im GDF-Format WGS 84 verwendet.

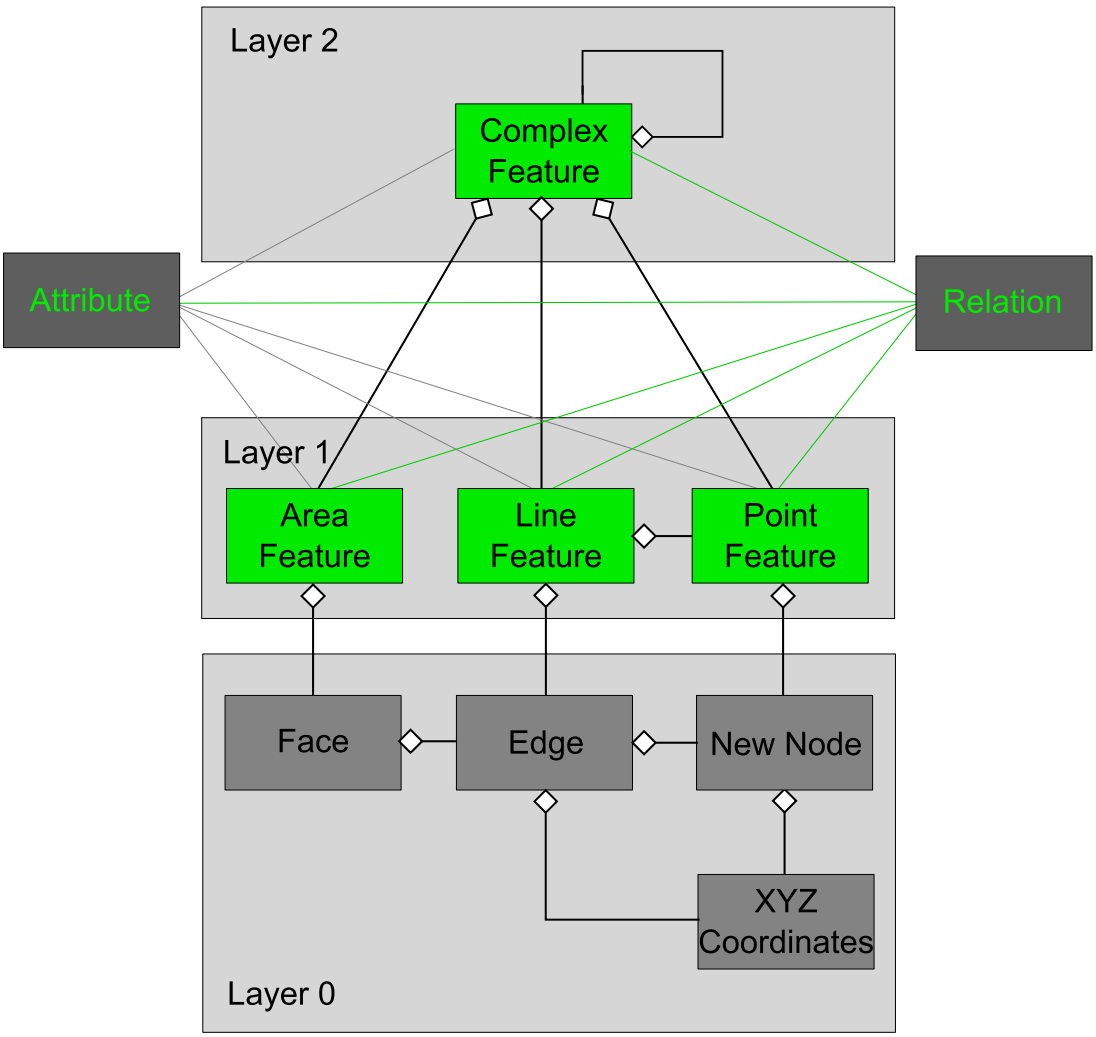


Abbildung 4.1.: GDF Datenmodell

Im **Layer 1** werden die Geometriedaten aus dem Layer 0 etwas weiter abstrahiert. Betrachtet man einen gewöhnlichen Atlas oder ein ähnliches Kartenwerk, so könnte man sagen dass diese im Grunde aus Punkten (**PointFeature**), Linien (**LineFeature**) und Flächen (**AreaFeature**) bestehen, die je nach ihrer Bedeutung ein etwas anderes Aussehen besitzen. So wird ein Wald beispielsweise als eine grüne oder ein See als eine blaue Fläche gezeichnet. Genau dies drücken auch die Elemente in diesem Layer aus. Mit Hilfe eines vierstelligen Zahlencodes wird den Elementen eines Grundtyps eine Bedeutung, für die Kartendarstellung, zugewiesen. So wird beispielsweise aus einem Punkt mit dem Code 7315 ein Restaurant, einer Linie mit dem Code 4110 eine

Straße oder aus einer Fläche mit dem Code 7120 ein Wald. Der dabei verwendete Code wird als **FeatureClassCode** (FCC) bezeichnet. Zudem können den Elementen in diesem Layer bereits Attribute zugeordnet werden. Es spielt bei der Kartendarstellung beispielsweise auch eine Rolle, ob es sich bei einer Straße um eine kleine Ortsstraße oder eine vierspurige Autobahn handelt. Die Darstellung von Straßen besitzt zudem noch eine weitere Besonderheit. Diese beginnen und enden im GDF-Format stets an einem Punkt, der als eine Kreuzung gekennzeichnet ist. Es lässt sich also aus den Informationen im Layer 1 ein Graph aus Knoten und Kanten ableiten, der für eine Routenberechnung verwendet werden kann.

Der **Layer 2** enthält die sogenannten **ComplexFeatures**. Auch wenn der verwendete Name vielleicht darauf hindeutet, sind diese nicht komplizierter als die übrigen Features. Es handelt sich lediglich um eine weitere Abstrahierungsebene. Grundsätzlich sind ComplexFeatures in der Lage Elemente zu gruppieren und diesen Attribute zuzuweisen, wofür es zwei wichtige Einsatzgebiete gibt.

- **Generalisierung des Straßennetzes**
  Betrachtet man beispielsweise Autobahnen in einem gewöhnlichen Straßenatlas, so bestehen diese aus mehreren Fahrbahnen und an den Verbindungsstellen existieren oft komplexe, kleeblattförmige Auf- und Abfahrten. Auf der Übersichtsseite werden diese jedoch als einfache Linien dargestellt, die in einem Punkt verbunden sind. Mit Hilfe von ComplexFeatures lassen sich die Bestandteile von Straßen und Kreuzungen zusammenfassen, was beispielsweise eine einfachere Darstellung oder Zielführung ermöglicht.

- **Darstellung von Verwaltungsstrukturen**
  Um Funktionen, wie eine Zieleingabe, zu realisieren muss die Verwaltungsstruktur eines Landes nachgebildet werden können. Normalerweise lässt sich diese als eine baumartige Struktur darstellen. So besteht ein Land beispielsweise aus Bundesländern, die wieder aus Kreisen bestehen und so weiter. Dies kann mit ComplexFeatures als Teil-Ganzes-Beziehung modelliert werden. Elemente von der Größe einer Stadt oder kleiner werden allerdings als AreaFeature dargestellt.

Zur vollständigen Beschreibung des Datenmodells fehlen nur noch die **Relationen**. Diese sind ein Mittel zur Darstellung von Beziehungen zwischen den Elementen in Layer 1 und 2. Mit Hilfe von Relationen kann beispielsweise ausgedrückt werden, dass sich eine Straße oder ein Sonderziel in einer Stadt befindet, was für eine Zieleingabe benötigt wird. Es lassen sich aber auch Sachverhalte, wie ein Abbiegeverbot von einer Straße in eine Andere, damit modellieren. Der Typ der Beziehung wird durch einen

vierstelliger Zahlencode beschrieben. Dieser bestimmt auch die Art und Reihenfolge der beteiligten Elemente.

4.2. Dateiformat

Das Austauschformat für GDF-Daten ist ein einfaches Textformat, das Felder fester Breite vorsieht und 80 Zeichen pro Zeile enthält. Listing 4.1 zeigt einen kurzen Ausschnitt einer solchen Datei. Diese besteht aus sogenannten Records. Ein Record beginnt mit einem zweistelligen Zahlencode, der dessen Typ angibt. Anschließend folgen die für diesen definierten Felder. Endet eine Zeile mit einer 1, so wird der Record in der nächsten Zeile fortgesetzt, wobei diese mit dem Code 00 beginnt. Ist das letzte Zeichen eine 0 so ist der Record komplett. Das Dokument beginnt mit einigen Informationen, die den Aufbau des Dokuments beschreiben. Records mit dem Code 03 (Zeile 5 bis 7) beschreiben beispielsweise die darin verwendeten Felder. Ein einfaches Beispiel für einen Datenrecord ist in Zeile 9 zu sehen. Der Code 23 sagt aus, dass es sich bei den folgenden Daten um einen XYZ-Record mit Koordinaten handelt. Die Zahl, die auf den Code folgt, stellt die Kennung des Datensatzes dar. Nach ein paar Eigenschaften die nicht verwendet werden und deshalb als Leerzeichen dargestellt sind, folgt die Anzahl der enthaltenen Koordinaten (12). Nun beginnt eine Liste mit jeweils einer Zahl für X-, Y- und Z-Koordinate. Diese wird, mit einer Eins am Ende der Zeile, mehrmals fortgesetzt und endet schließlich in Zeile 14, mit einer abschließenden Null.

```
1  01NAVTEQ            GDF      3.0 101031                1       1        1
2  00UTF-8          10000000217  1(C) NAVTEQ NORTH AMERICA, SV14.00,FV07.01
3  000,DV2.4.3,CT14:42,DVNLUXEMBOURG LU 10400,LN4LUGAA5T                  0
4  0201            0000000217                                             0
5  03ALBUM_ID 10N     OBL                          UNIQUE ALBUM IDENTIFIER 0
6  03AREA_ID  10N     <S>   1          2E32        AREA FEATURE IDENTIFIER 0
7  03ATT_DESC * G     <S>                          ATTRIBUTE DESCRIPTION   0
8  ...
9  23     11416          12    605429  4982929        0    605354  4982942   1
10 00         0    605275  4982956        0    605228  4982964        0       1
11 00    605187  4982971        0    605136  4982981        0    605094      1
12 00   4982994        0    605043  4983013        0    605003  4983034      1
13 00         0    604965  4983052        0    604922  4983070        0       1
14 00    604882  4983082        0                                           0
```

Listing 4.1: GDF - Textformat

Das GDF-Format ist zwar schon etwas älter, lässt sich aber sehr gut um benutzerdefinierte Funktionen erweitern. Neben der Möglichkeit neue FeatureClassCodes oder Attribute hinzuzufügen, können sogar völlig neue Records mit den dafür verwendeten Feldern definiert werden.

5. Spatialite

Die Datenbanklösung Spatialite hat für die vorliegende Arbeit eine wesentliche Bedeutung und konnte maßgeblich zu deren Erfolg beitragen. Sie wurde ursprünglich von Alessandro Furieri, für eine Applikation zur Verwaltung von Fahrplänen für Zug- und Buslinien entwickelt. Spatialite basiert auf der embedded Datenbank SQLite und erweitert diese um, in OpenGIS definierte, Funktionen zur Verarbeitung von Geoinformationen. Für eine Lösung der Aufgabenstellung war es nötig, die Verarbeitung von großen Datenmengen zu ermöglichen. Um dies möglichst performant zu realisieren, war ein grundlegendes Verständnis über die Arbeitsweise von Spatialite erforderlich.

In diesem Kapitel wird zunächst auf Eigenschaften und Besonderheiten von SQLite eingegangen. Anschließend werden einige, in OpenGIS definierte, Datentypen und Operationen vorgestellt. Nach einer Beschreibung, wie diese in Spatialite realisiert wurden, folgen einige Hinweise zu deren effektiven Einsatz.

5.1. SQLite

SQLite ist im Grunde eine Programmbibliothek, die es ermöglicht eine Datei als relationales Datenbanksystem zu verwenden. Die erste Version wurde im Jahre 2000, von Richard Hipp, für die Raketenabwehr auf einem Kriegsschiff entwickelt. Viele Entwickler erkannten schon bald den enormen Nutzen der kleinen (ca. 300kB), in C geschriebenen, Bibliothek. Denn viele Anwendungen sind in ihrer Wesensart Datenbanken oder können zumindest von einer Ablagemöglichkeit für Daten, in einem relationalen Schema, profitieren. Welcher Anwender würde jedoch für ein kleines Programm, das beispielsweise Adressen oder Kontobewegungen verwaltet, die Installation eines vollständigen Datenbanksystems, wie MySQL oder PostgreSQL, in Kauf nehmen. Oft werden Daten, die eigentlich relationalen Charakter haben, beispielsweise in Form von XML-Dateien abgelegt und müssen für effiziente Abfragen komplett in den Speicher geladen werden. Eine Alternative ist die Verwendung eines eigenen Binärformats, das schnelle Abfragen nach bestimmten Kriterien ermöglicht. Dies ist jedoch keine einfache Aufgabe

und übersteigt oft die Möglichkeiten der meisten Anwendungsprogrammierer. Mit Hilfe von SQLite können auch große Datenmengen, schnell und mit geringem Speicherbedarf, verarbeitet werden. Dies macht es besonders für Anwendungen auf Geräten mit begrenztem Speicher attraktiv. SQLite ist deshalb ein Bestandteil vieler embedded Plattformen[28]. Aber auch auf dem Desktop oder im Web ist der Siegeszug von SQLite ohne gleichen. Viele bekannte Firmen und Open Source Projekte verwenden das praktische Werkzeug bereits [29] oder unterstützen sogar dessen Weiterentwicklung. SQLite bietet Wrapper für die meisten Programmiersprachen und ist ein fester Bestandteil der Skriptsprachen Python (>=2.5) und PHP5. Die SQL-Schnittstelle von SQLite lässt kaum Wünsche offen. Es wird ein Großteil der im Standard SQL-92 definierten SQL-Syntax verstanden und mit Transaktionen, Views und Triggern stehen einige mächtige Werkzeuge zur Verfügung. Was SQLite jedoch völlig fehlt, sind die Befehle der sogenannten **Data Control Language** (DCL), die bei anderen Datenbanksystemen zur Rechteverwaltung eingesetzt wird.

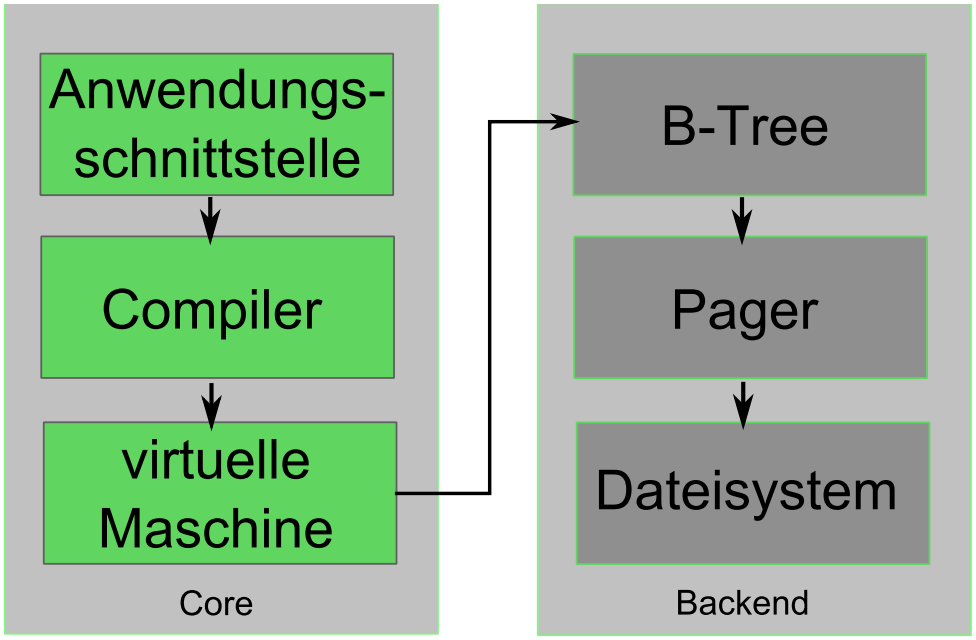


Abbildung 5.1.: Architektur von SQLite

Abbildung 5.1 zeigt den grundsätzlichen Aufbau von SQLite. Im Compiler werden SQL-Anweisungen, in Textform, in einen internen Bytecode übersetzt. Dieser wird dann von einer virtuellen Maschine ausgeführt, die 128 Opcodes für verschiedene Datenbankoperationen beherrscht. Die meisten SQL-Operationen sind als C-Funktionen reali-

[28] Android, Symbian, iPhone, Maemo, BlackBerry, MeeGo, webOS
[29] `http://www.hwaci.com/sw/sqlite/famous.html`

siert die, über einen Callback-Mechanismus, von der virtuellen Maschine aufgerufen werden. Eine SQLite-Datenbank wird auf der Platte mit Hilfe einer B-Tree Implementierung verwaltet. Ein B-Tree ist eine Datenstruktur, die schnelle Abfragen anhand eines Schlüssels ermöglicht. Für jede Tabelle und jeden Index in der Datenbank wird ein eigener B-Tree verwendet. Dieses Modul greift jedoch nicht direkt auf das Dateisystem zu, sondern über einen Zwischenspeicher den ein sogenannter Pager verwaltet. Die Daten in der Datei werden durch diesen in Blöcken fester Größe gelesen und geschreiben. Durch eine intelligente Verwaltung des Zwischenspeichers, wird die Anzahl der Dateizugriffe auf ein Minimum reduziert. Eine Konfiguration des Pager-Moduls ermöglicht eine Skalierung der SQLite-Datenbank für verschiedene Anwendungsbereiche, wobei die Größe und die Anzahl der verwendeten Blöcke eingestellt werden können. SQLite verwendet eine Standardeinstellung mit 20 000 Blöcken zu je 1024 Byte, was einen Speicherbedarf von sparsamen 20 MB ergibt. Im Rahmen dieser Arbeit wurde die Blockgröße auf 32 kB geändert, was den Zwischenspeicher auf maximal 640 MB erhöht. Dadurch konnte die Verarbeitung von großen Datenmengen erheblich beschleunigt werden.

5.2. OpenGIS

„Ein **Geoinformationssystem** (GIS) ist ein datenverarbeitungsgestütztes Informationssystem zur Erfassung, Verwaltung, Analyse, Modellierung und Visualisierung von Geodaten mit dem Ziel der Gewinnung von Geoinformationen.“ [Streit, 2011, Kap. 9.1]

Das **Open Geospatial Consortium** (OGC) legt mit **OpenGIS** einen Standard für die Interoperabilität von GIS-Systemen fest. Das Dokument `OpenGIS® Simple Features Specifications For SQL`[30] beschreibt eine Umgebung die sich `SQL with Geometry Types` nennt und von Spatialite unterstützt wird. Es werden darin einige geometrische Datentypen und Operationen definiert, von denen hier nur einige vorgestellt werden.

Abbildung 5.2 zeigt die Struktur der, in OpenGIS definierten, Datentypen. Die Elemente `MultiPoint` und `MultiLineString` wurden weggelassen, da sie in dieser Arbeit nicht verwendet werden. Es mag vielleicht etwas seltsam erscheinen, dass diese als eine Vererbungshierarchie, in einem ein UML-Klassenmodell, beschrieben werden. Bekanntlich gibt es in SQL keine Klassen und es ist dadurch weder Vererbung noch

[30] [OGC, 1999]

Komposition möglich. Dennoch ist dieses Modell sehr hilfreich. Das Prinzip der Vererbung ist es, strukturelle oder verhaltensbezogene Gemeinsamkeiten von Objekten zusammenzufassen. Eine Klasse definiert demnach Operationen, die auf eine Reihe von ähnlichen Objekten angewendet werden können. Die Klasse `Geometry` fasst hier alle Operationen zusammen, die sich auf grundsätzlich alle Geometrien anwenden lassen. Bei jeder Ableitung kann dabei eine Spezialisierung in Form von weiteren Operationen erfolgen. So ist eine Flächenberechnung nur bei Geometrien sinnvoll, die eine Fläche beschreiben (`Surface`). Ebenso ist es bei der Vererbung möglich das Verhalten der, für die Basisklasse definierten, Operationen zu verändern. Ist das Verhalten einer Operation der Basisklasse von der verwendeten Unterklasse abhängig, so wird dies als Polymorphie bezeichnet.

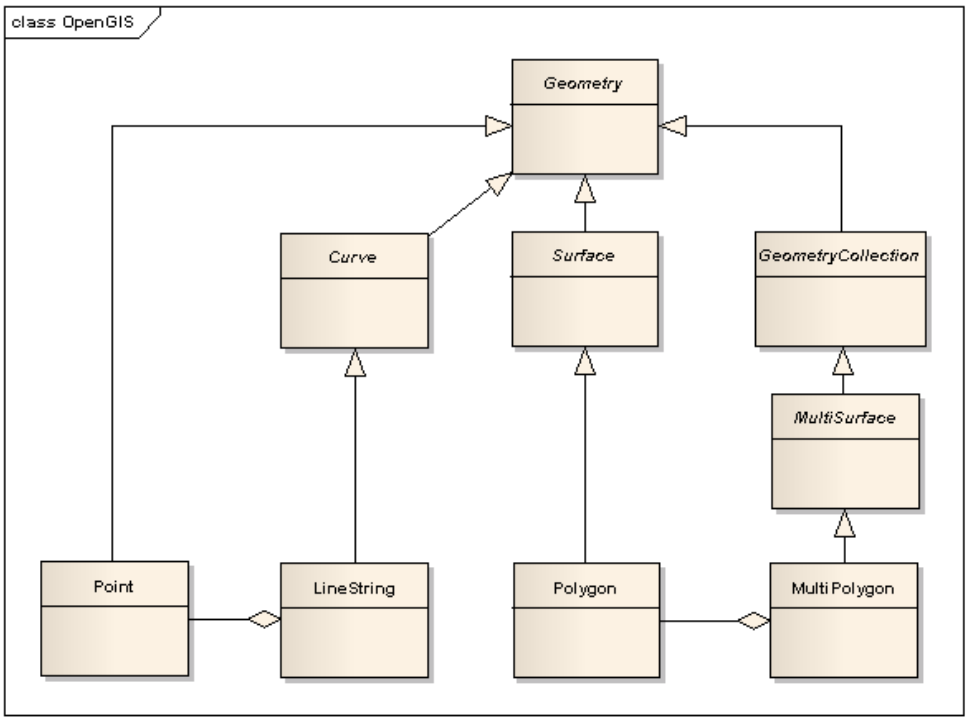


Abbildung 5.2.: OpenGIS-Datentypen (vereinfacht)

Im Grunde sind Klassen aber nur ein Werkzeug um Daten mit den darauf möglichen Operationen zu verknüpfen. Fehlt in einer Programmiersprache dieses Stilmittel, so definiert man einfach einen abstrakten Datentyp mit einer Reihe von Operationen, die auf diesen angewendet werden können. In einer SQL-Datenbank können beliebige binäre Daten als **Binary Large Object** (BLOB) gespeichert werden. Ruft man eine SQL-Funktion mit den Daten in einem BLOB auf, so entscheidet diese als was die darin abgelegten Daten interpretiert werden. Gibt es in den Daten ein Erkennungsmerkmal, das auf die verwendete Unterklasse hindeutet, so kann hier die Verarbei-

tung an eine dafür spezialisierte Funktion weitergeleitet werden, um ein polymorphes Verhalten nachzubilden. Für den Austausch, über Datenbankgrenzen hinweg, werden in OpenGIS die beiden Formate **Well-known Binary** (WKB) und **Well-known Text** (WKT) definiert. Für die Erzeugung der in dieser Arbeit benötigten Geometrien wurde das WKT-Format verwendet, für das einige Beispiele in Tabelle 5.1 aufgelistet sind[31].

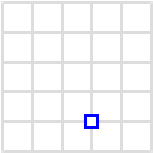
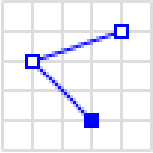
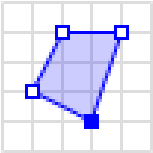
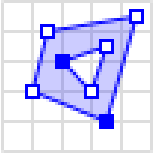
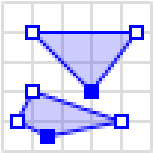
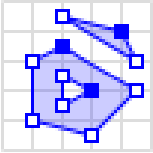
Geometrietyp	WKT-Repräsentation	Darstellung
Point	'POINT (30 10)'	
LineString	'LINESTRING (30 10, 10 30, 40 40)'	
Polygon	'POLYGON ((30 10, 10 20, 20 40, 40 40, 30 10))'	
Polygon	'POLYGON ((35 10, 10 20, 15 40, 45 45, 35 10), (20 30, 35 35, 30 20, 20 30))'	
MultiPolygon	'MULTIPOLYGON (((30 20, 10 40, 45 40, 30 20)), ((15 5, 40 10, 10 20, 5 10, 15 5)))'	
MultiPolygon	'MULTIPOLYGON (((40 40, 20 45, 45 30, 40 40)), ((20 35, 45 20, 30 5, 10 10, 10 30, 20 35),(30 20, 20 25, 20 15, 30 20)))'	

Tabelle 5.1.: Well Known Text - Format

In Tabelle 5.2 werden einige wichtige, in OpenGIS definierte, Operationen aufgelistet. Neben Funktionen die eine Konvertierung zwischen dem intern verwendeten BLOB-Format und einer Darstellung als WKT oder WKB ermöglichen, gibt es Operationen um räumliche Beziehungen zwischen Geometrien auswerten zu können. Die Berech-

[31] [Wikipedia, 2011, Well-known text]

nung von Flächen, Abständen oder Längen ist ebenso möglich, wie das Erzeugen neuer Geometrien mit Hilfe von Mengenoperationen. Zudem gibt es noch einige Methoden um den Zugriff auf die Bestandteile von zusammengesetzten Geometrien wie `LineString` oder `MultiPolygon` zu ermöglichen.

Funktion	Rückgabe/Beschreibung
GeomFromText(wkt String [,SRID Integer]) : Geometry	erzeugt Geometrie aus WKT
AsText(g1 Geometry) : String	liefert WKT-Darstellung der Geometrie
Equals(g1 Geometry, g2 Geometry) : Integer	**TRUE**, wenn g1 = g2
Disjoint(g1 Geometry, g2 Geometry) : Integer	**TRUE**, wenn es keine Schnittmenge von g1 und g2 gibt
Touches(g1 Geometry, g2 Geometry) : Integer	**TRUE**, wenn g1 und g2 sich berühren
Within(g1 Geometry, g2 Geometry) : Integer	**TRUE**, wenn g1 komplett in g2 enthalten
Overlaps(g1 Geometry, g2 Geometry) : Integer	**TRUE**, wenn g1 und g2 sich überlappen
Crosses(g1 Geometry, g2 Geometry) : Integer	**TRUE**, wenn Schnittmenge kleinere Dimension als g1 und g2 hat
Intersects(g1 Geometry, g2 Geometry) : Integer	**TRUE**, wenn eine Schnittmenge existiert
Area(s Surface) : Double	ermittelt die Größe der übergebenen Fläche
Distance(g1 Geometry, g2 Geometry) : Double	ermittelt den geringsten Abstand zwischen g1 und g2
Intersection(g1 Geometry, g2 Geometry) : Geometry	liefert die Schnittmenge aus g1 und g2
Difference(g1 Geometry, g2 Geometry) : Geometry	liefert den Unterschied zwischen g1 und g2

Tabelle 5.2.: Auszug aus den, in OpenGIS definierten, SQL - Funktionen

In OpenGIS werden zusätzlich noch einige Tabellen definiert, in denen Metadaten über, in der Datenbank enthalte, Geodaten verwaltet werden. Die Tabelle *spatial_ref_sys* enthält Informationen zu verwendeten Koordinatensytemen mit den für eine Umrechnung benötigten Parametern. In *geometry_columns* befindet sich eine Liste aller Tabellenspalten, die Geometriedaten enthalten. Zudem wird hier der verwendete Geometrietyp, Indizierung, Anzahl der Dimensionen und das verwendete Koordinatensystem angegeben. Damit sich der Benutzer nicht selbst um die Aktualisierung dieser Daten kümmern muss, gibt es einige nützliche Befehle, die dies automatisieren (Tabelle 5.3).

Aufruf	**Funktion**
InitSpatialMetadata(void) : Integer	erzeugt die Tabellen *geometry_columns* und *spatial_ref_sys*
AddGeometryColumn(table String, column String, srid Integer, geom_type String, dimension Integer) : Integer	Fügt einer Tabelle eine Geometriespalte hinzu und aktualisiert die Metadaten
CreateSpatialIndex(table String, column String) : Integer	erzeugt einen RTRee-Index und markiert die Spalte in den Metadaten als Indiziert

Tabelle 5.3.: zusätzliche SQL-Funktionen von Spatialite

Mit Hilfe dieser Metadaten weiß ein GIS-System, wie die Daten zu interpretieren sind und kann diese zur Anzeige bringen. Neben, dem zu Spatialite gehörenden, Spatialite-GIS können die Daten beispielsweise auch durch QGIS[32] oder OpenJUMP[33] verarbeitet werden.

[32] http://www.qgis.org
[33] http://www.openjump.org

5.3. Aufbau von Spatialite

Abbildung 5.3 zeigt den grundsätzlichen Aufbau von Spatialite. Um die von OpenGIS geforderte Funktionalität zu realisieren wurde der SQL-Befehlssatz von SQLite, durch Spatialite, erweitert. Hierfür steht in SQLite die sogenannte SQL-Extension zur Verfügung. Mit Hilfe der Funktion `sqlite3_create_function` (Listing 5.3) können einfach neue, in C geschriebene, Funktionen bei der virtuellen Maschine registriert werden.

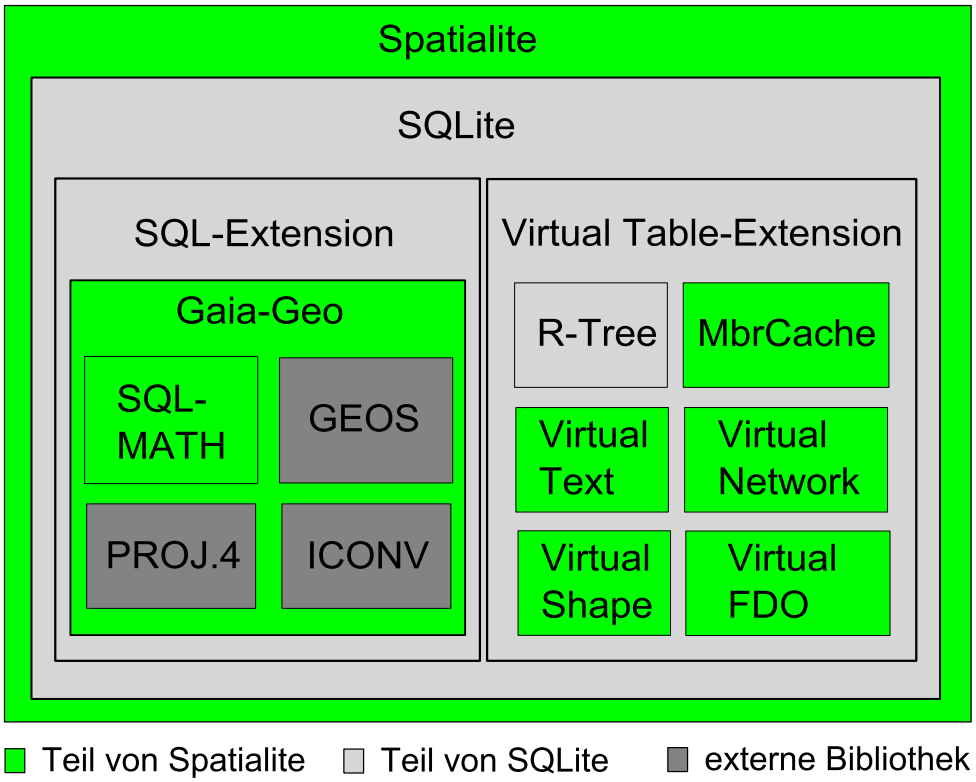


Abbildung 5.3.: Aufbau von Spatialite

```
int sqlite3_create_function (
    sqlite3 *db,  // Zeiger auf Datenbankinstanz
    const char *zFunctionName, // Name der SQL-Funktion
    int nArg,     // Anzahl der Parameter
    int eTextRep, // Zeichensatz
    void *pApp,   // Zeiger auf Anwendungsdaten (meist nicht verwendet)
    void (*xFunc)(sqlite3_context*,int,sqlite3_value**), // Funktionszeiger
    void (*xStep)(sqlite3_context*,int,sqlite3_value**),
    void (*xFinal)(sqlite3_context*)
);
```

Listing 5.1: Schnittstelle von `sqlite3_create_function`

Listing 5.2 zeigt ein einfaches Beispiel für die Funktion `ABS`, die den Absolutwert des übergebenen Wertes zurückliefert und in Spatialite implementiert ist. Meist wird nur der Funktionszeiger `xFunc` benötigt. `xStep` und `xFinal` können für Aggregatsfunktionen ähnlich `MAX`, `MIN` oder `AVG` benutzt werden, was bei Spatialite allerdings nicht verwendet wird. Mehr zu diesem Thema ist z. B. in [Owens, 2006] zu lesen.

```
1  static void
2  fnct_math_abs (sqlite3_context * context, int argc, sqlite3_value ** argv)
3  {
4      sqlite3_int64 int_value;
5      double x;
6      if (sqlite3_value_type (argv[0]) == SQLITE_FLOAT)
7      {
8          x = fabs (sqlite3_value_double (argv[0]));
9          sqlite3_result_double (context, x);
10     }
11     else if (sqlite3_value_type (argv[0]) == SQLITE_INTEGER)
12     {
13         int_value = math_llabs (sqlite3_value_int64 (argv[0]));
14         sqlite3_result_int64 (context, int_value);
15     }
16     else
17         sqlite3_result_null (context);
18 }
19
20 sqlite3_create_function (db, "abs", 1, SQLITE_ANY, 0, fnct_math_abs, 0, 0);
```

Listing 5.2: Hinzufügen der Funktion `ABS` zum SQLite-Befehlssatz

Das Beispiel zeigt, wie einfach auf die übergebenen Parameter zugegriffen und das Ergebnis an den aufrufenden Kontex zurückgeliefert werden kann. Anstelle einer Berechnung in C, können in der benutzerdefinierten Funktion natürlich auch SQL-Statements ausgeführt werden. Ein gutes Beispiel hierfür ist die Spatialite-Funktion `CreateSpatialIndex(tabelle,spalte)`. Diese legt für die übergebene Tabellenspalte einige Trigger an, die einen R-Tree-Index aktualisieren. Ein R-Tree ist, ähnlich dem B-Tree, eine Datenstruktur die schnelle Abfragen ermöglicht. Für eine Abfrage können hier jedoch die Koordinaten eines Rechtecks verwendet werden, in dem die Daten enthalten sein sollen. Als Zwischenschicht zur SQL-Schnittstelle existiert in Spatialite eine C-API namens gaia, die grundsätlich die selbe Funktionalität wie die SQL-Schnittstelle implementiert. Eine der wichtigsten Aufgaben von gaia ist es eine C-kompatible Schnittstelle zur objektorientierten GEOS-Bibliothek[34] herzustellen, deren Algorithmen für die meis-

[34] `http://trac.osgeo.org/geos/`

ten geometrischen Operationen verwendet werden. Außerdem übernimmt sie die Konvertierung von SQLite-BLOB zum WKT bzw. WKB-Format und zurück und implementiert einige einfache Algorithmen selbst. Die Bibliothek PROJ.4[35] wird lediglich für den SQL-Befehl `Transform` verwendet, welcher eine übergebene Geometrie in ein anderes Koordinatensystem überführt. Die ICONV-Bibliothek[36] wird in keinem der neuen SQL-Befehle verwendet, sondern dient als Unterstützung für das VirtualText-Modul. Dort ist sie dafür zuständig den in der Textdatei verwendeten Zeichensatz in das SQLite-interne UTF-8 zu übersetzen.

Zusätzlich bietet Spatialite noch einige Erweiterungen, die über die sogenannte Virtual Table-Extension von SQLite realisiert wurden. Über diese Schnittstelle lassen sich unterschiedliche Datenquellen als sogenannte virtuelle Tabellen einbinden. Auf die Daten kann dann mit normalen SQL-Befehlen zugegriffen werden. Über das schon angesprochene Virtual Text-Modul lassen sich dadurch CSV-Dateien und über Virtual Shape sogenannte Shapefiles[37], ein sehr populäres Austauschformat für Geodaten das auf dBASE basiert, verwenden. Mit dem Modul Virtual Network können Straßendaten, die beispielsweise aus einem Shapefile stammen, zu einer einfachen Routenberechnung verwendet werden. Virtual **Feature Data Objects** (FDO) ermöglicht einen Zugriff auf Geodaten aus verschiedensten Quellen. FDO definiert hierzu eine einheitliche API, die die Form der Datenspeicherung abstrahiert[38]. Mit dem MbrCache existiert eine Alternative zum R-Tree, die ähnliche Funktionalität bietet. Dieser war jedoch durch einen enormen Speicherbedarf von etwa 40 MB je 1 000 000 indizierter Elemente für die vorliegende Arbeit uninteressant. Das R-Tree-Modul ist ebenfalls als virtuelle Tabelle realisiert, gehört jedoch zu den Bestandteilen von SQLite. Mit Ausnahme des R-Trees, auf dessen Besonderheiten im nächsten Abschnitt noch eingegangen wird, wurden keine virtuellen Tabellen zur Lösung der Aufgabe verwendet. Deshalb wird, an dieser Stelle, nicht weiter auf diese eingegangen [39].

[35] `proj.osgeo.org`
[36] `http://www.gnu.org/s/libiconv/`
[37] `http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf`
[38] `http://fdo.osgeo.org/`
[39] Eine etwas genauere Beschreibung ist auf der beiliegenden CD enthalten.

5.4. Besonderheiten bei der Verwendung

Für die Verarbeitung von großen Datenmengen in Spatialite (bzw. SQLite) im Allgemeinen und speziell für Geodaten müssen einige Besonderheiten beachtet werden, auf die hier kurz eingegangen werden soll. Dieser Abschnitt richtet sich an Leser, die bereits einige grundlegende Kenntnisse über die Arbeitsweise von Datenbanken besitzen. Alle Angaben für SQLite gelten für Spatialite analog.

Allgemeine Besonderheiten

Ab einer gewissen Größe von Tabellen ist die Verwendung von Indizes ein Muss. Im Zusammenhang mit SQLite gibt es dabei einige Besonderheiten:

- **Nur ein Index pro Tabelle kann verwendet werden**
  SQLite kann für eine Abfrage nur einen Index pro Tabelle verwenden. Müssen in einer Abfrage mehrere Spalten einer Tabelle ausgewertet werden, so kann es sinnvoll sein hierfür einen Index über mehrere Spalten zu erzeugen.
- **Primärschlüssel**
  Für Primärschlüssel wird in SQLite ein automatischer Index erzeugt. Wird jedoch `AUTOINCREMENT` verwendet, so entfällt dieser. SQLite verwaltet die Inhalte von Tabellen intern in einer bereits indizierten Struktur. Hierfür gibt es eine spezielle Spalte mit dem Namen `ROWID`, die für jede Tabelle existiert. Da die mit `AUTOINCREMENT` erzeugten Schlüssel mit der `ROWID` übereinstimmen, wird kein zusätzlicher Index benötigt. Die höchste `ROWID` aller Tabellen mit `AUTOINCREMENT` befindet sich in einer speziellen Tabelle Namens `sqlite_sequence`. Abfragen der Form `SELECT MAX(id) FROM tabelle` sind dadurch besonders effizient.

Zudem besitzt SQLite keine Query-Optimierung. Bei einer Abfrage über mehrere Tabellen hat die Reihenfolge der abgefragten Tabellen eine Auswirkung auf die Geschwindigkeit der Abfrage. Eine Vorgehensweise, die meist gute Ergebnisse liefert, ist es die Tabelle mit dem härtesten Ausschlusskriterium zuerst, in der `FROM`-Klausel, zu nennen.

Hierbei kann die Anweisung `EXPLAIN QUERY PLAN` eine Hilfe sein, die Informationen über die geplante Ausführung eines Querys liefert (Listing 5.3).

```
EXPLAIN QUERY PLAN
SELECT * FROM OsmNodes WHERE OsmId IN (SELECT node FROM (
SELECT node, COUNT(node) AS count
FROM OsmWay_Nodes, OsmTags
WHERE way=ref AND key='highway' AND type=1
GROUP BY node HAVING count > 1 ))
```

Listing 5.3: Query Plan

In Abbildung 5.4 ist die zugehörige Ausgabe zu sehen. Diese liefert die Reihenfolge der im Query verwendeten Tabellen und Informationen über die hierfür verwendeten Indizes. Mehr Informationen über die Verwendung dieses nützlichen Hilfsmittels sind unter [40] zu finden.

	order	from	detail
1	0	0	TABLE OsmNodes WITH INDEX sqlite_autoindex_OsmNodes_1
2	0	0	TABLE OsmWay_Nodes WITH INDEX idx_OsmWay_Nodes_Node ORDER BY
3	1	1	TABLE OsmTags WITH INDEX idx_OsmTags_ref_type

Abbildung 5.4.: Ergebnis der QueryPlan - Abfrage

Verarbeitung von Geodaten

Für die Verarbeitung von Geodaten werden mehrdimensionale Indizes benötigt. Diese Funktionalität wird vom R-Tree-Modul bereitgestellt, das eine Indizierung mit bis zu fünf Dimensionen ermöglicht. Da dieses aber kein fester Bestandteil von SQLite ist, sondern über die Virtual Table-Extension realisiert wurde, muss der Index, bei einer Abfrage, explizit angegeben werden. Um dies zu veranschaulichen, soll ein einfaches Beispiel dienen. Abbildung 5.5 zeigt die Ausgangssituation für eine einfache geometrische Abfrage. Es sollen dabei die Linien, die ganz oder teilweise in einer Fläche enthalten sind, abgefragt werden.

[40] http://www.sqlite.org/eqp.html

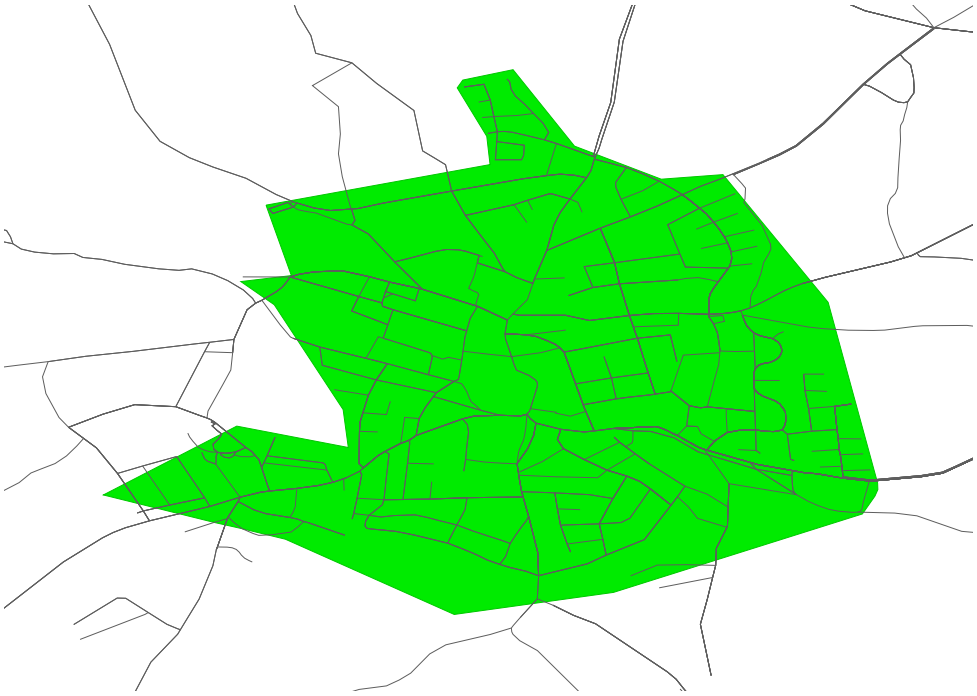


Abbildung 5.5.: Linien und Flächen in Spatialite-GIS

Listing 5.4 zeigt eine einfache Abfrage, die dies bewerkstelligt. Die Operation `Intersects` wird dabei auf alle Datensätze in der Tabelle mit den Linien angewendet. Da diese aufwendige Berechnungen durchführt, dauert die Abfrage sehr lange. Für Tabellen mit mehreren millionen Einträgen ist diese Vorgehensweise nich sinnvoll.

```
1 SELECT Linien.id FROM Flaechen, Linien WHERE Flaechen.id=1
2 AND Intersects(Linien.geom, Flaechen.geom);
```

Listing 5.4: geometrische Abfrage ohne Index

Listing 5.5 verwendet einen R-Tree zur Vorfilterung der Daten, auf Basis des umgebenden Rechtecks der Fläche. Dadurch ist diese um einiges schneller als die Abfrage in Listing 5.4. Leider sind hierfür verschachtelte `SELECT`s erforderlich.

```
SELECT 1 AS flaechenId, Linien.id FROM Flaechen, Linien WHERE
Flaechen.id=flaechenId
AND Flaechen.ROWID IN (
    SELECT pkid FROM idx_Flaechen_geom, Flaechen WHERE
     Flaechen.id=flaechenId AND
     xmin > MbrMinX(Flaechen.geom) AND
     xmax < MbrMaxX(Flaechen.geom) AND
     ymin > MbrMinY(Flaechen.geom) AND
     ymax < MbrMaxY(Flaechen.geom)
)
AND Intersects(Linien.geom, Flaechen.geom);
```

Listing 5.5: geometrische Abfrage mit Index

Die Erzeugung eines R-Tree-Index kann für eine beliebige Tabellenspalte, die eine Geometrie enthält, mit Hilfe der Operation `CreateSpatialIndex` erfolgen (Listing 5.6).

```
SELECT CreateSpatialIndex('Linien', 'geom');
```

Listing 5.6: Erzeugen eines räumlichen Index

Diese legt eine virtuelle Tabelle, unter Verwendung des R-Tree-Moduls, an und erzeugt einige Trigger, die sich um deren Aktualisierung kümmern. In Listing 5.7 ist ein, auf diese Weise erzeugter, `INSERT`-Trigger zu sehen.

```
CREATE TRIGGER "gii_Linien_geom" AFTER INSERT ON "Linien"
FOR EACH ROW BEGIN
INSERT INTO "idx_Linien_geom" (pkid, xmin, xmax, ymin, ymax) VALUES (
NEW.ROWID,
MbrMinX(NEW."geom"), MbrMaxX(NEW."geom"), MbrMinY(NEW."geom"), MbrMaxY(NEW."geom"));
END
```

Listing 5.7: Von *CreateSpatialIndex* erzeugter INSERT-Trigger

6. Bearbeitung der Aufgabenstellung

Die hier vorgestellte Lösung zur Konvertierung von OpenStreetMap-Kartendaten in das kommerzielle PSF-Format für Navigationslösungen auf Basis des **EB street director SDK** der Firma Elektrobit wurde mit Hilfe von Visual Studio 2008 unter Windows XP in der Programmiersprache C++ entwickelt. Als Datenbank kam Spatialite zum Einsatz, eine Erweiterung der bekannten SQLite-Datenbank um OpenGIS-kompatible Geometriedatentypen. Nach Möglichkeit wurde die EB-eigene PICommon-Bibliothek verwendet. Dies ist eine plattformunabhängige Sammlung von Klassen, die einige nützliche Datentypen und Container bereitstellt.

Dieses Kapitel beginnt mit einer Beschreibung des Gesamtkonzepts der bearbeiteten Aufgabe, gefolgt von einer Übersicht über den Aufbau der gewählten Lösung. Nach einer Beschreibung der Implementierung einiger relevanter Bestandteile werden die Ergebnisse der Arbeit vorgestellt.

6.1. Konzept

Bei EB stehen die Kartendaten verschiedener Anbieter im GDF-Format zur Verfügung. Diese werden zuerst, durch den GDFReader, in eine objektrelationale, auf Gigabase[41] basierende Datenbank eingelesen, bevor sie durch den PSFCreator weiterverarbeitet werden. Deshalb ist es nicht zwingend notwendig GDF-Dateien zu erzeugen um OpenStreetMap-Daten verwenden zu können. Da die Daten durch den GDFReader bereits um einige Zusatzinformationen erweitert werden, wurde jedoch darauf verzichtet die GDF-Datenbank direkt zu generieren. Das in Abbildung 6.1 dargestellte Konzept versucht, bereits bestehenden Code des GDFReaders optimal zu nutzen.

Dazu wird das Datenmodell von OpenStreetMap und GDF in einer, auf Spatialite basierenden, Datenbank nachgebildet. Die zu entwickelnde Applikation (OSMReader) hat die Aufgabe, OpenStreetMap-Daten einzulesen und in das GDF-Datenmodell zu

[41] `http://www.garret.ru/gigabase.html`

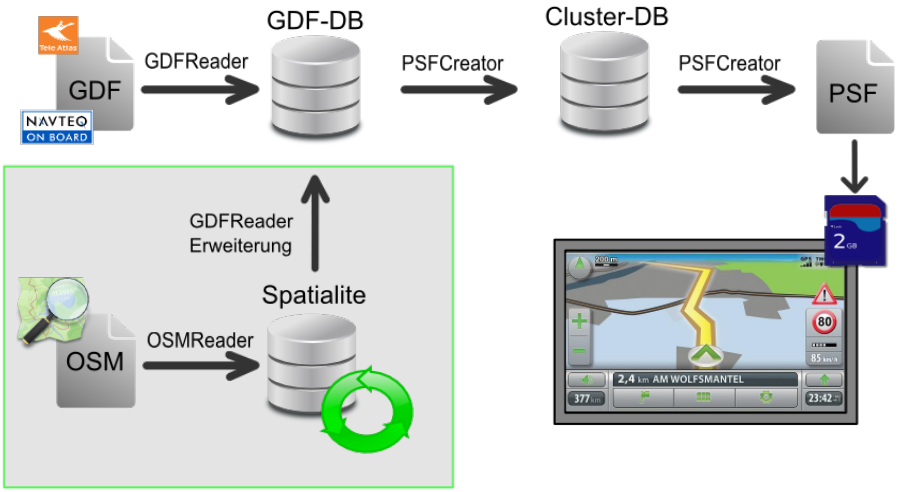


Abbildung 6.1.: Konzept für den zu entwickelnden Konverter

überführen (Abbildung 6.2). Diese Nachbildung des GDF-Formates in einer relationalen Datenbank wurde als **Elektrobit relational GDF** (ERG) bezeichnet. Mit Hilfe einer geeigneten Erweiterung, importiert der GDFReader die so entstandenen Daten und generiert daraus die GDF-Datenbank.

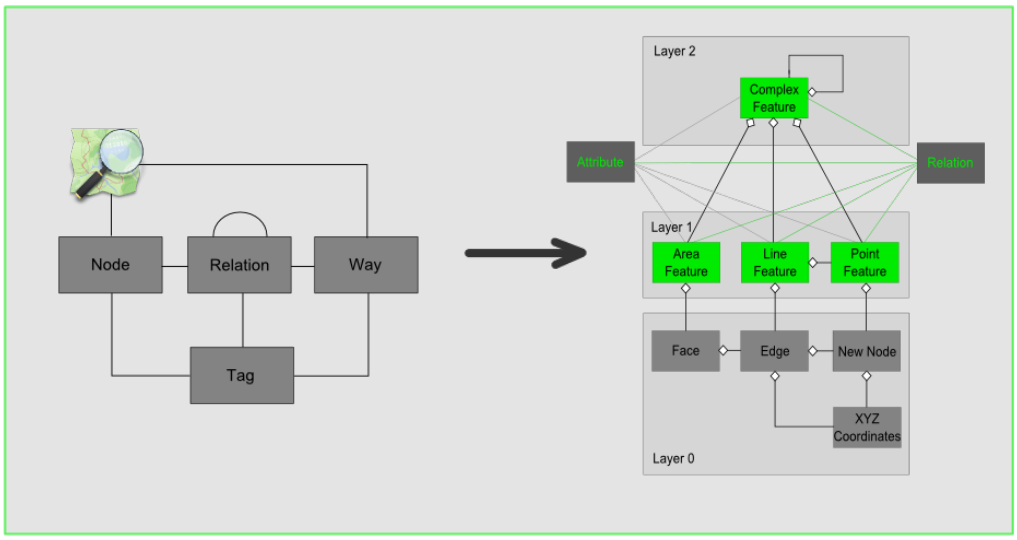


Abbildung 6.2.: Konzept für den OSMReader

6.2. Vorgehensweise

Angesichts des Umfangs der im GDF-Format beschriebenen Features und Attribute war schon sehr früh abzusehen, dass eine Berücksichtigung aller Details in der zur Verfügung stehenden Zeit nicht erreicht werden kann. Nach einer Analyse der grundlegenden Anforderungen, an den zu entwerfenden Konverter, wurde deshalb ein iterativer Ansatz zur Lösung des Problems gewählt. Die Anforderungen wurden dazu in kleine Teile zerlegt, die innerhalb weniger Tage realisiert werden konnten. Aus einer Anforderung wie „Nodes und Ways aus OSM können auf Point-, Line- und Areafeatures in GDF abgebildet werden“ wurde beispielsweise die Teilaufgabe „Eine Node, die im OSM-Format ein Restaurant darstellt, soll in das GDF-Format übertragen und im MapViewer dargestellt werden können“ abgeleitet. Die einzige Bedingung an den erstellten Code war, dass spätestens am Ende einer Woche ein funktionierendes und prüfbares Teilergebnis existiert. Nach dem Lösen einiger solcher Teilaufgaben wurde der Code restrukturiert. Dabei kamen einige Gedanken aus dem Buch Clean Code[42] zum Einsatz. Hier einige Dinge, die dabei beachtet wurden:

- verwende immer die einfachste Lösung
- vermeide Wiederholungen
- eine Klasse - eine Aufgabe
- verwende Komposition um kompliziertere Probleme zu lösen
- programmiere Nichts für die Zukunft
- Kommentare werden, wenn möglich, durch sprechende Funktionsnamen ersetzt
- Funktionen sollen nicht länger als 30 Zeilen sein
- Funktionen sollen auf einer logischen Ebene operieren
- Funktionen sollen möglichst wenige Parameter besitzen

Mit Hilfe von Visual Assist[43] wurden hierzu einige einfache Refactoring-Muster angewendet. Anfangs wurden nur sehr kleine Kartenausschnitte aus OpenStreetMap verarbeitet. So konnten die Auswirkungen eines veränderten Code-Bestandteils innerhalb

[42] [Martin, 2009]
[43] `http://www.wholetomato.com`

weniger Sekunden überprüft werden. Interessanterweise wird bei dieser Vorgehensweise das Hinzufügen neuer Features immer einfacher und schneller. Durch die ständige Arbeit mit dem Code bekommt man ein Gefühl für besonders änderungsanfällige Bestandteile und oft benötigte Hilfsmittel. Von Zeit zu Zeit wurden, mit Hilfe des Enterprise Architect[44], UML-Diagramme aus vorhandenem Code erzeugt, um einen besseren Überblick über das Gesamtsystem zu erhalten. Bei der Verarbeitung großer Datenmengen traten oft Engpässe bei der Verarbeitungsgeschwindigkeit auf. Durch die Verwendung des Tools VerySleepy[45], als Profiler, konnten die verantwortlichen Code-Bestandteile sehr schnell bestimmt und die Probleme beseitigt werden. Große Änderungen am Code waren dadurch selten nötig. Für die statische Codeanalyse wurde PCLint[46] eingesetzt. Der Code ist dabei, bis auf wenige Ausnahmen, konform.

6.3. Übersicht

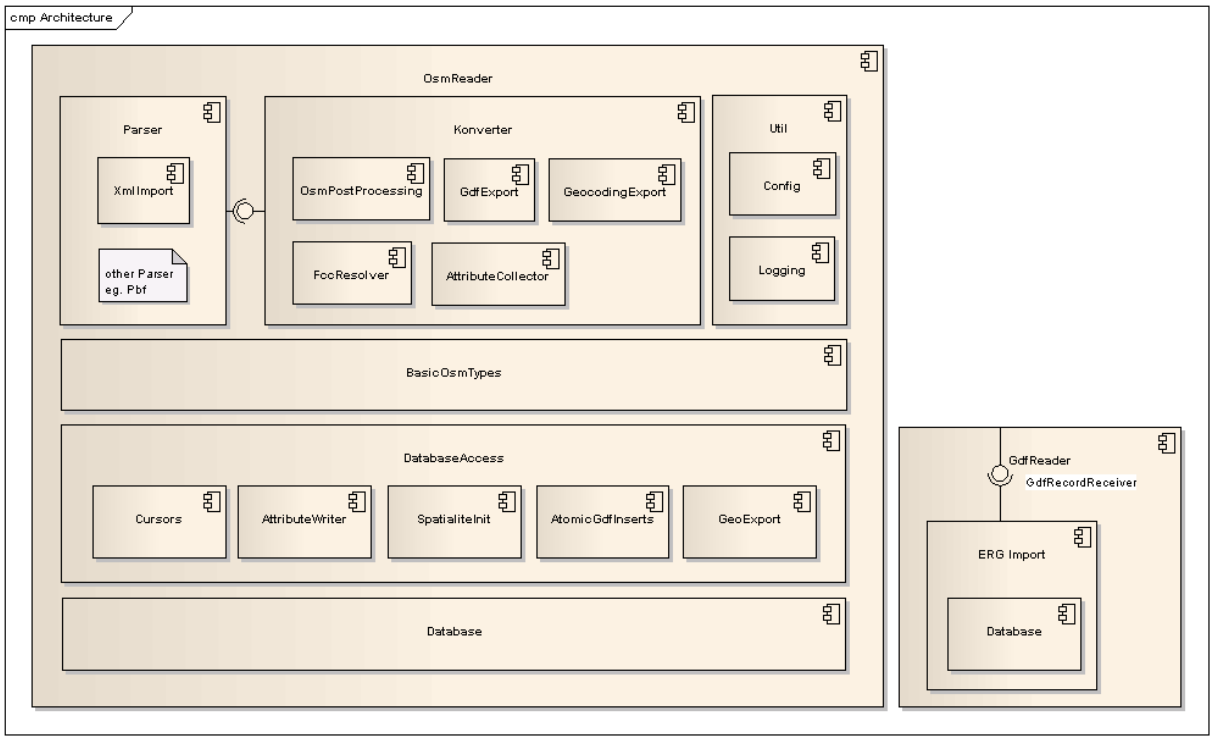


Abbildung 6.3.: Übersicht zum entwickelten Konverter

[44] www.sparxsystems.de
[45] http://www.codersnotes.com/sleepy
[46] http://www.gimpel.com

Abbildung 6.3 soll eine grobe Übersicht über den entstandenen Konverter bieten. Im Grunde besteht die Applikation aus zwei Teilen. Einen Konverter, der OSM-Daten in das GDF-Datenmodell überführt und eine Erweiterung des GDFReaders zum Import der generierten Daten. Auf die Spatialite-Datenbank (Database) wird, bei beiden Teilen, auf unterster Ebene mit CppSQLite3[47] zugegriffen, einen schmalen Wrapper um die C-API von SQLite für C++. Der GDFReader-Teil ist schnell erklärt. Hier wird eine intern bereits vorhandene Schnittstelle für den Import der generierten Daten verwendet. Auf die Datenbank wird dazu direkt zugegriffen. Im Konverter, der den Namen OSMReader trägt, wird der Zugriff auf die Datenbank und somit fast der gesamte SQL-Code in wenigen Klassen gekapselt (DatabaseAccess). Das Klassenmodell für die Elemente aus OpenStreetMap (BasicOsmTypes) hat sich als sehr flexibel erwiesen und kommt, vor allem im Zusammenhang mit dem Cursor-Konzept, in fast allen Teilen des Konverters zum Einsatz. Zum Einlesen der OSM-Daten im XML-Format ist ein Parser vorhanden (XmlImport). Eine einfache Schnittstelle ermöglicht die Erweiterung um andere Formate (z. B. OsmBin oder PBF). Der eigentliche Konverter besteht nur aus einer Hand voll einfacher Module:

- **OsmPostProcessing**
  hier werden den OSM-Daten Informationen hinzugefügt, die sich direkt aus diesen ableiten lassen
- **FccResolver**
  erfüllt die wichtige Aufgabe einem Element aus OpenStreetMap, anhand des Grundtyps und verwendeter Tags, einem FeatureClassCode in GDF zuzuordnen
- **AttributeCollector**
  erzeugt GDF-Attribute aus OSM-Tags mit Hilfe des, durch den FccResolver ermittelten, FeatureClassCodes
- **GdfExport**
  exportiert die GDF-Grundelemente Point-, Line- und AreaFeature. Dabei werden für einige Features Geometriedaten erzeugt, die später für das Geocoding verwendet werden können
- **GeocodingExport**
  erzeugt aus Geometriedaten Relationen, die für die Zieleingabe benötigt werden

[47] `http://www.codeproject.com/KB/database/CppSQLite.aspx`

Zusätzlich gibt es noch eine einfache Konfiguration (Config) und die Möglichkeit zur Erstellung eines Logfiles (Logging). Es gibt auch bereits einige elementare UnitTests, die sich bei Bedarf erweitern lassen. Das zugehörige Datenmodell (Abbildung 6.4) besteht aus drei logischen Einheiten:

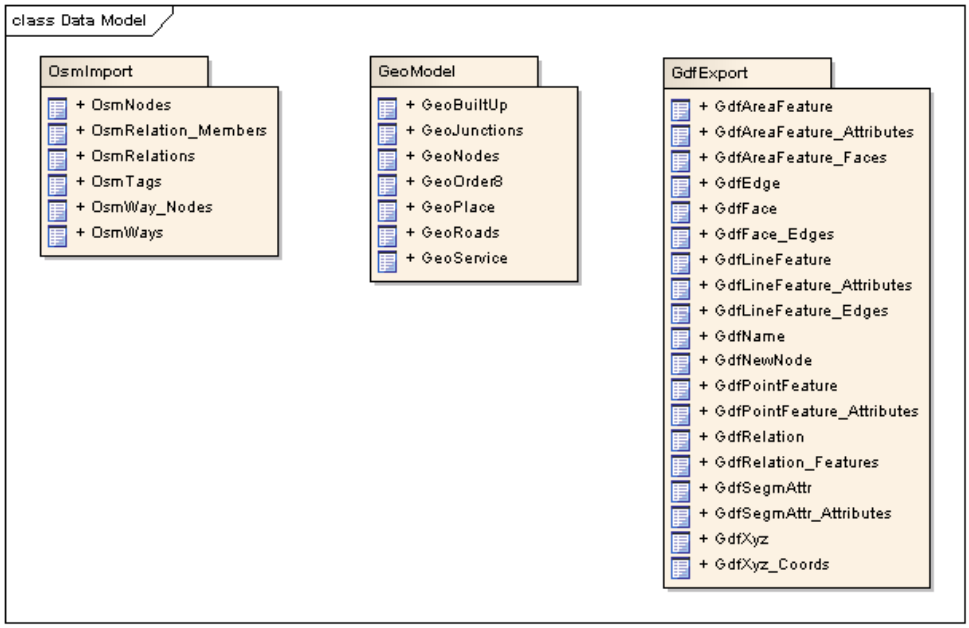


Abbildung 6.4.: Datenmodell

- **OsmImport**
  repräsentiert die aus OpenStreetMap eingelesenen Daten
- **GdfExport**
  ist ein relationales Abbild des GDF-Datenmodells
- **GeoModel**
  enthält Informationen über Geometrien einiger erzeugter GDF-Elemente aus denen sich topologische Informationen ableiten lassen, die beispielsweise zur Zieleingabe benötigt werden

6.4. Implementierung

In diesem Abschnitt wird die Implementierung einiger relervanter Bestandteile des Konverters beschrieben. Die verwendeten UML-Diagramme, Illustrationen und Quellcodeauszüge sollen dem Leser helfen die jeweiligen Problemstellungen und deren Lösung leicht zu verstehen und sind aus diesem Grund teilweise vereinfacht dargestellt.

6.4.1. Erweiterung des GDFReaders

Der GDFReader bietet intern eine Schnittstelle mit dem Namen `GDFRecordReceiver`, die durch die Klasse `GDFRecordReceiverImpl` implementiert wird. In der Methode `ParseFile` registriert sich diese als Empfänger für die von einem GDF-Parser gelesenen Datensätze, die als Records bezeichnet werden, und startet diesen. In der Realisierung der, in der Schnittstelle definierten, Methoden befinden sich die Einfügeoperationen für die GDF-Datenbank. Durch Überschreiben der Methode `ParseFile` wird der GDF-Parser durch die Klasse `ErgImporter` ersetzt, welche die Datensätze nun aus der Spatialite-Datenbank liest (Abbildung 6.5). Änderungen an Konfiguration und Benutzerschnittstelle fallen gering aus und werden nicht näher beschrieben.

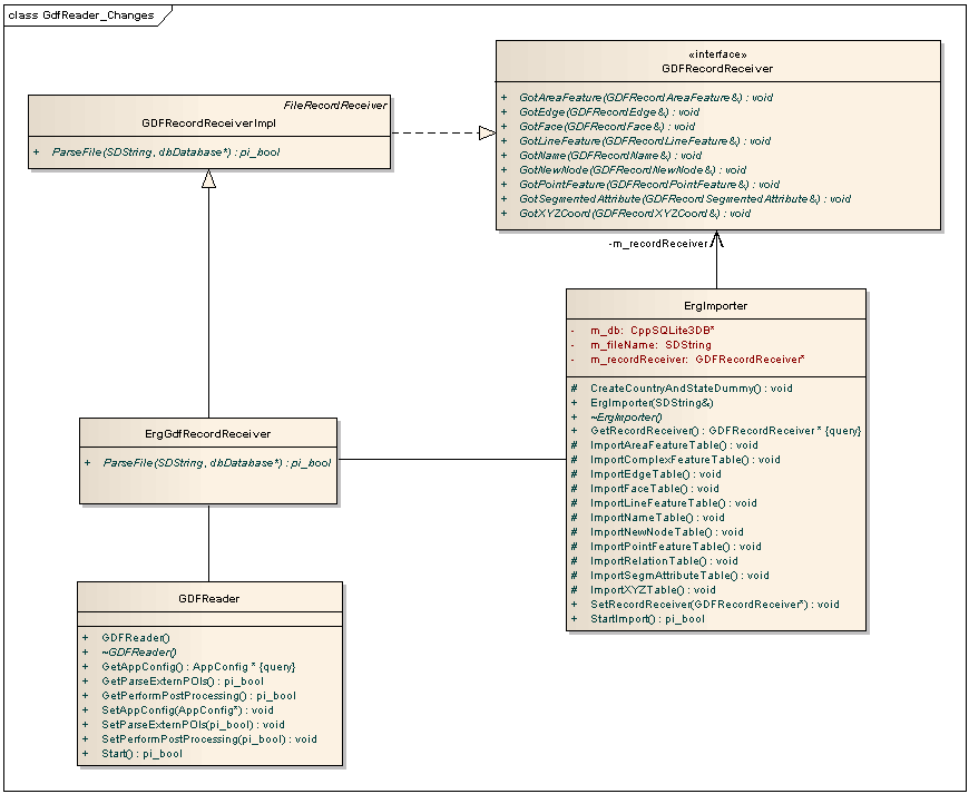


Abbildung 6.5.: Erweiterung des GDFReaders

6.4.2. Einlesen der OSM-Daten

Für den Import der OpenStreetMap-XML-Daten wird ein ein einfacher Parser nach dem Prinzip der **Simple API for XML** (SAX) verwendet. Ein SAX-Parser liest ein XML-Dokument sequentiell. Spezielle Zeichenfolgen wie z. B. die Start- oder Endmarkierung eines Elements, werden dabei als Ereignis aufgefasst. Über die Schnittstelle `SAXContentHandler` können für solche Ereignisse Behandlungsroutinen registriert werden, in denen die so gewonnenen Daten weiterverarbeitet werden. Da ein SAX-Parser kein Gedächtnis benötigt, ist der Speicherbedarf unabhängig von der Größe des einzulesenden Dokuments und fällt sehr gering aus. Dies ist wichtig, da hier XML-Dokumente in einer Größenordnung von mehrenen Gigabyte verarbeitet werden müssen. Konzepte, die XML-Daten vollständig in den Speicher einlesen, wie das **Document Object Model** (DOM) können aus diesem Grund nicht verwendet werden.

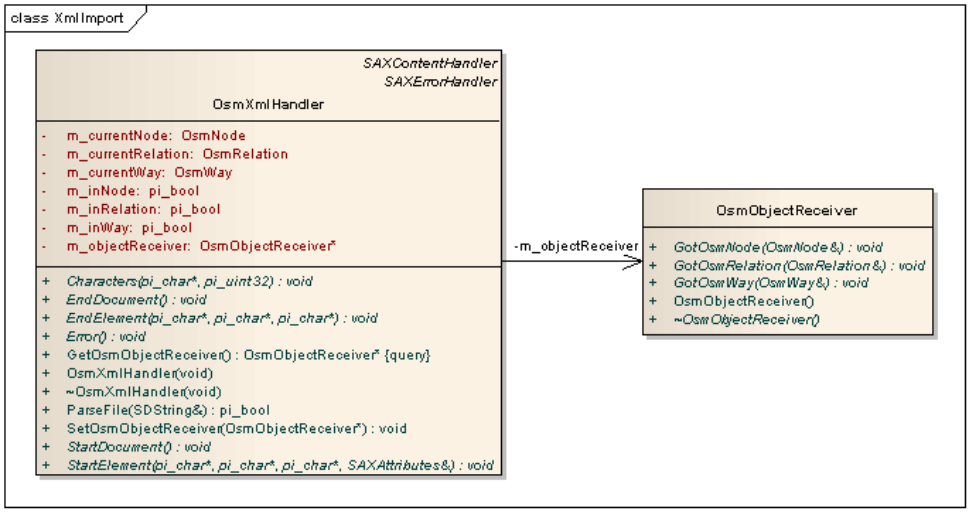


Abbildung 6.6.: XML-Import

Abbildung 6.6 zeigt die Realisierung des XML-Imports. `OsmXmlHandler` implementiert die beschriebenen Ereignis-Behandlungsroutinen, registriert sich in der Methode `ParseFile` beim SAX-Parser und startet diesen. Die Schnittstelle `OsmObjectReceiver` nimmt die aus den XML-Daten erzeugten Objekte entgegen.

6.4.3. Klassenmodell für die OSM-Daten

Das Klassenmodell für die OpenStreetMap-Daten ist leicht nachzuvollziehen (Abbildung 6.7). Jedes Element in OSM besitzt eine eindeutige Kennung und kann optional eine beliebige Menge von Tags besitzen. Diese gemeinsamen Eigenschaften werden in der abstrakten Basisklasse `OsmObject` zusammengefasst. Die abstrakte Methode `GetOsmType` muss von den konkreten Unterklassen implementiert werden. Wird als Parameter `withArea` der Wert `true` verwendet, so wird bei Ways die eine Fläche beschreiben `OSM_AREA` zurückgeliefert. Ohne Parameter gibt es als Rückgabewerte nur `OSM_NODE`, `OSM_WAY` und `OSM_RELATION`. Die virtuelle Methode `IsRotated` wird von `OsmWay` überschrieben und liefert `true`, wenn dessen Richtung umgekehrt wurde. `OsmNode` erweitert die Basisklasse um Informationen zur geografischen Länge und Breite und `OsmWay` um eine geordnete Liste aus Referenzen auf Nodes. `OsmRelation` enthält zusätzlich Informationen zu ihren Bestandteilen, die aus Typ, Role und Referenz bestehen.

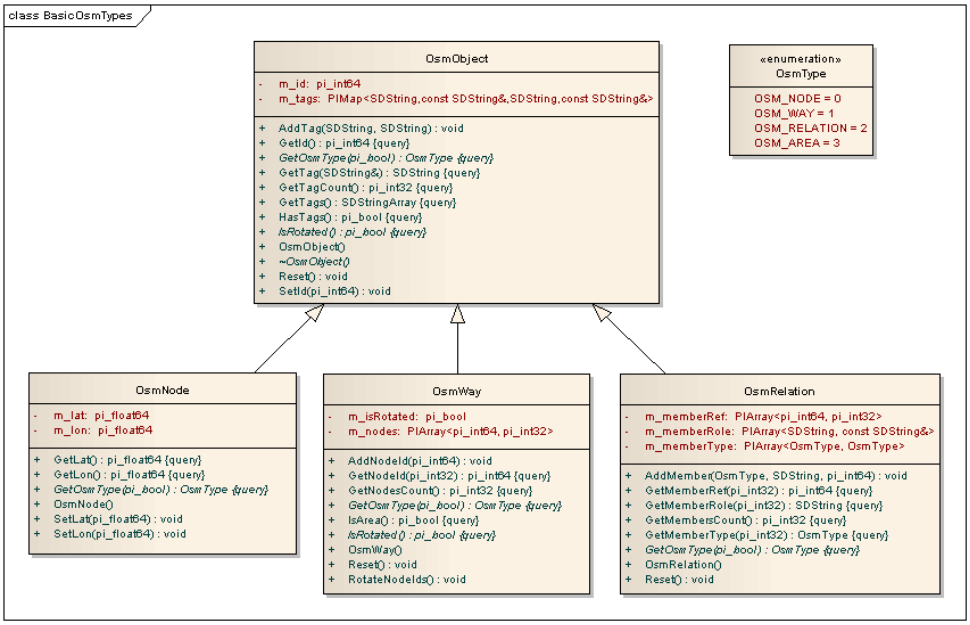


Abbildung 6.7.: OSM Basisklassen

6.4.4. Datenmodell für die OSM-Daten

Das Datenmodell für die OpenStreetMap-Elemente in Abbildung 6.8 besteht grundsätzlich aus einer Tabelle für jedes OSM-Grundelement. Für Elemente, die aus mehreren Bestandteilen bestehen, sind Zuordnungstabellen entstanden. Da die Kennungen in OpenStreetMap nur für jeweils ein Grundelement eindeutig sind, wurde in der Tabelle für die Tags ein zusätzliches Typ-Feld eingeführt. Hierdurch genügt eine einzelne Tabelle für alle vier Grundelemente.

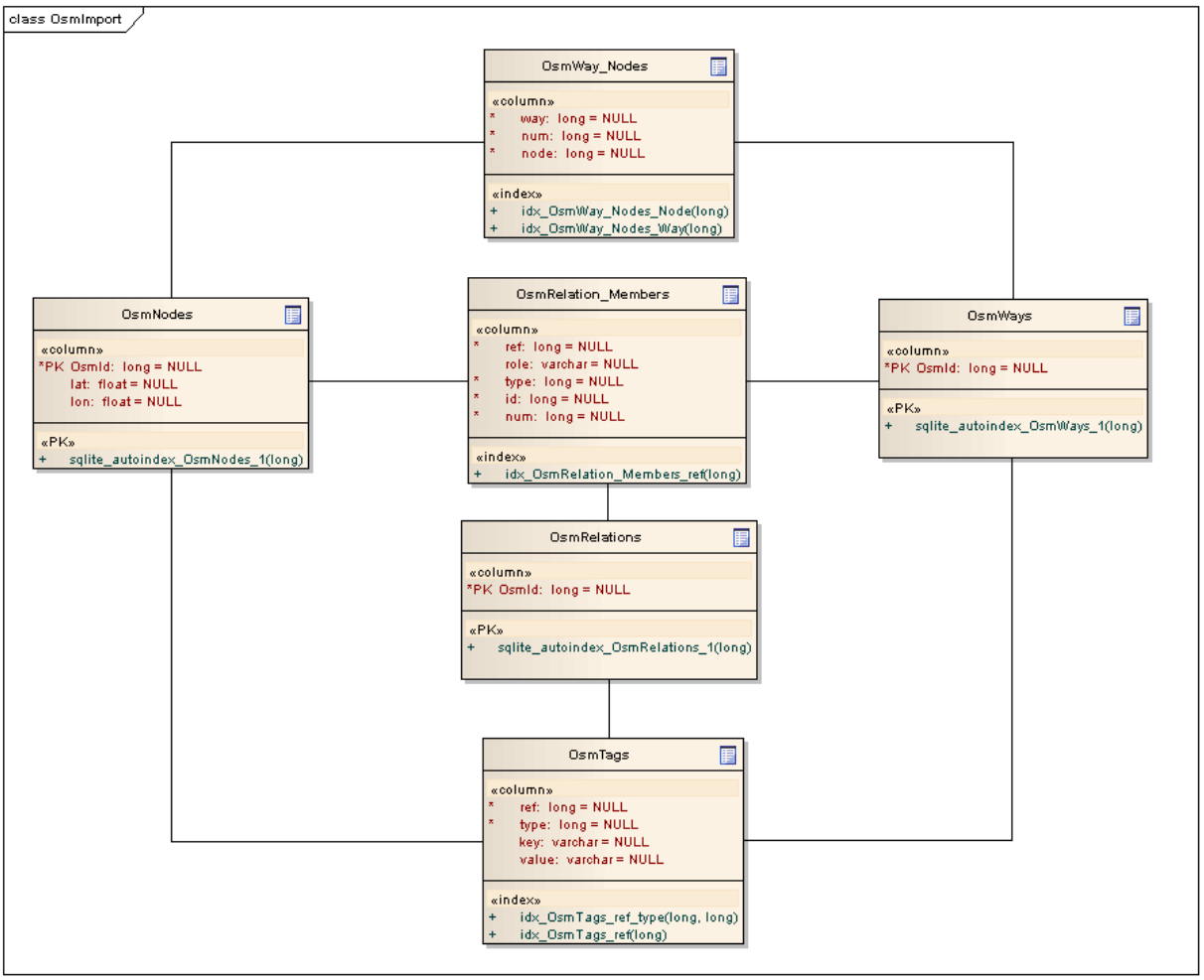


Abbildung 6.8.: Datenmodell für den OSM-Import

6.4.5. Cursor-Konzept

Um komfortabel mit den OpenStreetMap-Daten arbeiten zu können, wird für diese eine einfache Zwischenschicht implementiert. Die Anwendungsfälle hierfür sind:

- einfügen von OSM-Objekten in die Datenbank
- durchlaufen einer selektierten Menge eines OSM-Grundtyps
- modifizieren einzelner Elemente

Dies lässt sich am einfachsten mit Hilfe eines, hier als Cursor bezeichneten, Konzepts erreichen. Mit Operationen, die an einen für die Programmiersprache JAVA üblichen Iterator angelehnt sind, kann das Ergebnis einer Abfrage durchlaufen werden. Dabei stehen die Ergebnisse bereits als Objekte zur Verfügung. Kenntnisse über das verwendete Datenmodell sind somit nicht mehr erforderlich.

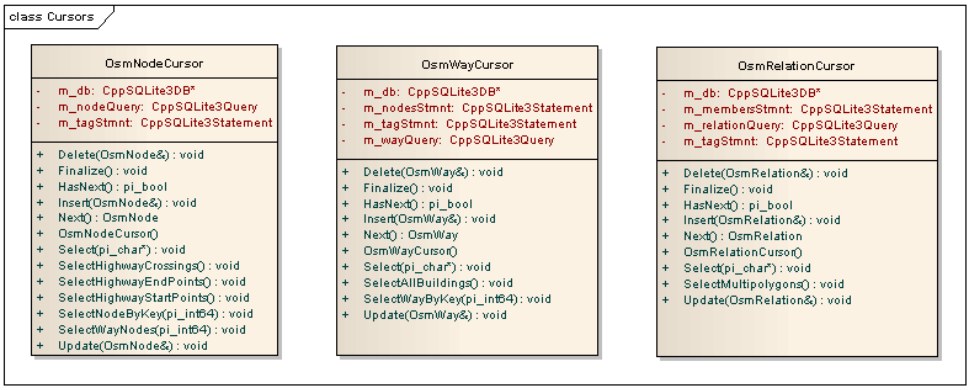


Abbildung 6.9.: Cursor für die OSM-Grundelemente

Listing 6.1 zeigt ein einfaches Beispiel zur Verwendung des Cursors für den Typ `OsmNode`. Hierbei wird eine selektierte Menge von Nodes durchlaufen. Nodes, die nicht als Kreuzung markiert sind, werden als solche gekennzeichnet und in der Datenbank aktualisiert.

```
OsmNodeCursor crossingsCursor;
crossingsCursor.SelectHighwayCrossings();

while ( crossingsCursor.HasNext() )
{
    OsmNode node = crossingsCursor.Next();
    if (node.GetTag("junction") != "")
    {
        node.AddTag("junction", "yes");
        crossingsCursor.Update(node);
    }
}
```

Listing 6.1: Verwendung von `OsmNodeCursor`

6.4.6. Vorverarbeitung der OSM-Daten

Im Modul OsmPostProcessing werden den OSM-Daten Informationen hinzugefügt, die sich direkt aus diesen ableiten lassen (Abbildung 6.10).

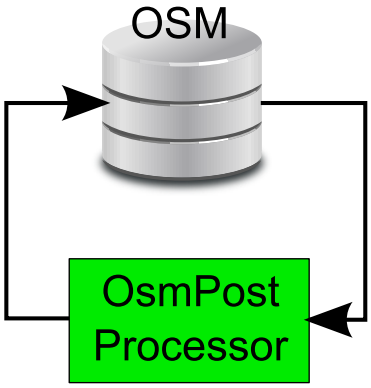


Abbildung 6.10.: OsmPostProcessor

Im Moment erfüllt das Modul zwei wichtige Aufgaben, zu denen die Kennzeichnung von Kreuzungen und das Generieren von POIs aus Gebäuden gehört.

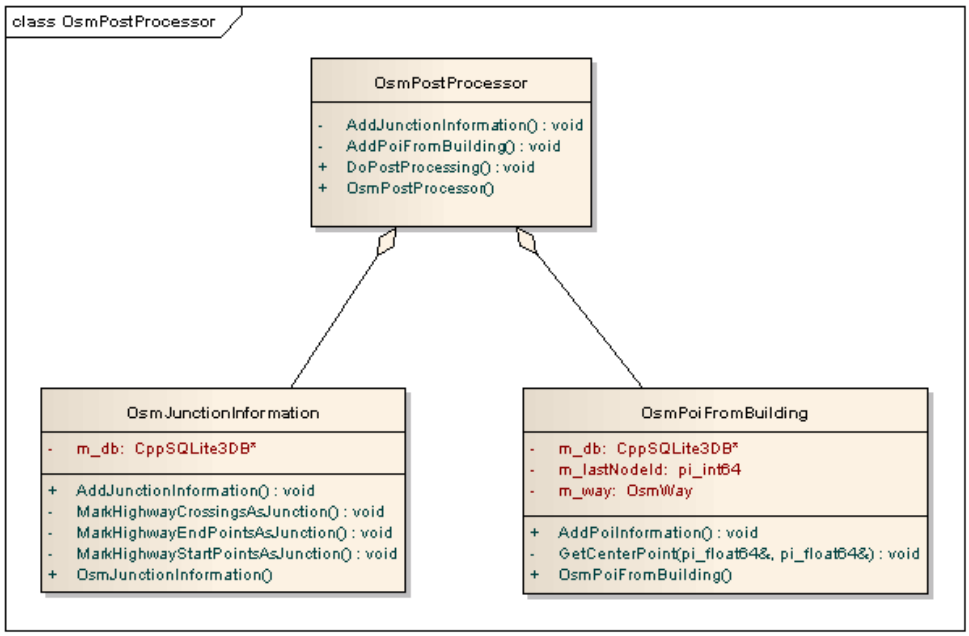


Abbildung 6.11.: OsmPostProcessor - Klassen

Kennzeichnung von Kreuzungen und Wegenden

In OpenStreetMap spielt es keine Rolle ob eine Verbindung von zwei Straßen als Kreuzung markiert ist oder nicht. Zum Rendern von Kartenkacheln werden meist nur sehr kleine Kartenausschnitte verwendet, die komplett in den Speicher geladen werden können. Dadurch kann diese Information leicht ermittelt werden. Im GDF-Format müssen Straßen jedoch als einfache Kanten ohne Abzweige dargestellt werden, die an einem Knoten beginnen und auch wieder an einem Knoten enden. Der Grund dafür ist leicht nachvollziehbar, denn die Daten sollen zur Routenberechnung genutzt werden können. Dafür kommen Algorithmen wie A* oder der Djikstra-Algorithmus zum Einsatz, die alle auf Graphen, mit oben beschriebenem Aufbau, operieren.

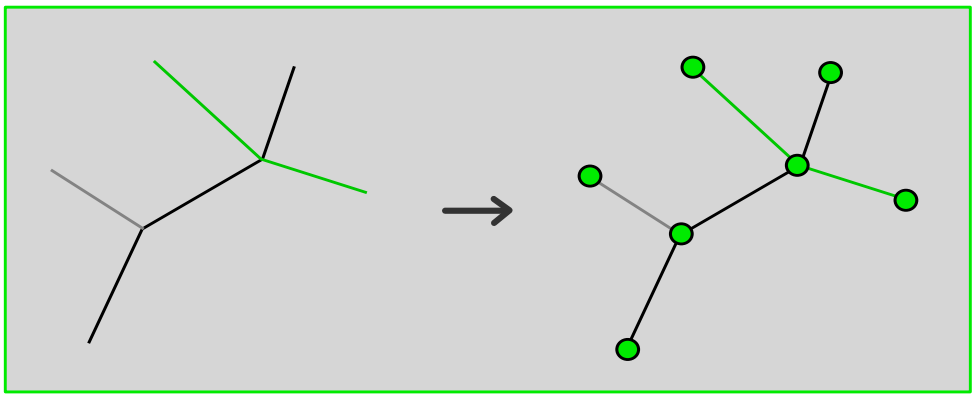


Abbildung 6.12.: Kennzeichnung von Kreuzungen und Wegenden

Die Klasse `OsmJunctionInformation` übernimmt die in Abbildung 6.12 dargestellte Kennzeichnung der Kreuzungen und Wegenden, die später zum Auftrennen der zugehörigen Wege verwendet werden wird. Interessanterweise ist es mit dem verwendeten Datenmodell eine Kleinigkeit, die benötigte Information zu erzeugen. Es sind nur drei, sehr einfache, Abfragen auszuführen.

Die erste Abfrage ermittelt Nodes, die sich in mehr als einem als Straße gekennzeichneten Way befinden. Handelt es sich dabei nicht bereits um eine Kreuzung, so wird diese als Kreuzung markiert. Listing 6.2 zeigt, wie einfach sich dies mit der Verwendung eines Cursors realisieren lässt.

```
OsmNodeCursor crossingsCursor;
crossingsCursor.SelectHighwayCrossings();

while ( crossingsCursor.HasNext() )
{
    OsmNode node = crossingsCursor.Next();
```

```
7       if (node.GetTag("junction") != "")
8       {
9           node.AddTag("junction", "yes");
10          crossingsCursor.Update(node);
11      }
12  }
```

Listing 6.2: Markierung von Kreuzungen mit dem `OsmNodeCursor`

Die beiden weiteren Abfragen bestimmen die erste und letzte Node in allen als Straße gekennzeichneten Ways und verfahren mit diesen ähnlich. Durch die Verwendung geeigneter Indizes ist diese Lösung nicht nur unglaublich einfach, sondern auch sehr schnell.

Generieren von POIs oder Adressen aus Gebäuden

In OpenStreetMap können Sonderziele oder einfache Hausadressen auch als Fläche dargestellt werden (meist Gebäude). Im GDF-Format sind hierfür aber ausschließlich Punkte definiert. Gebäude können zwar dargestellt, aber bei der Zieleingabe nicht berücksichtigt werden. Es müssen also aus einigen Flächen Punkte mit identischen Tags generiert werden. Die vorerst gewählte Lösung ist, das kleinste Rechteck das die gesamte Fläche eines Gebäudes enthält und dessen Mittelpunkt zu ermitteln (Abbildung 6.13). Dies liefert in den meisten Fällen gute Ergebnisse, kann aber unter Umständen einen Punkt außerhalb der Fläche generieren.

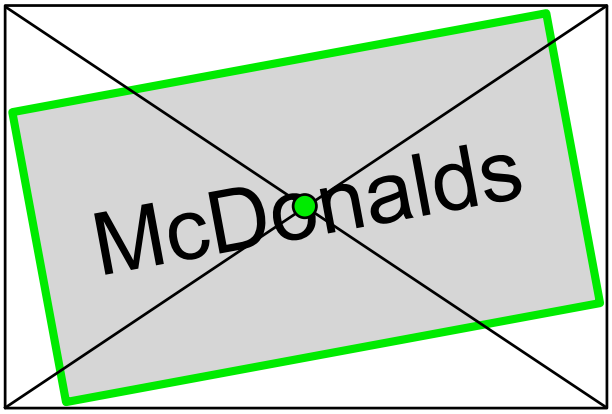


Abbildung 6.13.: Punkt aus Polygon

```
void OsmPoiFromBuilding::GetCenterPoint( pi_float64 &lat, pi_float64 &lon )
{
    OsmNodeCursor nodeCursor;
    OsmNode node;
    pi_bool firstRun = TRUE;
    pi_float64 minLat, maxLat, minLon, maxLon;

    nodeCursor.SelectWayNodes(m_way.GetId());

    while (nodeCursor.HasNext())
    {
        node = nodeCursor.Next();
        lat = node.GetLat();
        lon = node.GetLon();

        if (firstRun)
        {
            minLat = maxLat = lat;
            minLon = maxLon = lon;
            firstRun = FALSE;
        }

        minLat = (lat < minLat)?lat:minLat;
        maxLat = (lat > maxLat)?lat:maxLat;
        minLon = (lon < minLon)?lon:minLon;
        maxLon = (lon > maxLon)?lon:maxLon;
    }

    lat = (maxLat - minLat) / 2.0 + minLat;
    lon = (maxLon - minLon) / 2.0 + minLon;
}
```

Listing 6.3: Punkt aus Polygon

In der GEOS-Bibliothek ist ein etwas aufwendigerer Algorithmus vorhanden, der dies berücksichtigt und später von dort übernommen werden kann.

6.4.7. Ermittlung des FeatureClassCodes

Ein **FeatureClassCode** (FCC) ist ein, in der GDF-Spezifikation festgelegter, vierstelliger Zahlencode. Dieser Zahlencode hat in Verbindung mit einem der Grundelemente Point-, Line-, Area- oder ComplexFeature eine spezielle Bedeutung. So handelt es sich beispielsweise bei einem LineFeature mit dem FeatureClassCode 4110 um eine Straße. Es ist nicht schwer Objekte aus OpenStreetMap einem Grundelement in GDF zuzuordnen, denn Punkte, Linien und Flächen gibt es hier ebenfalls. Um was es sich dabei wirklich handelt z. B. eine Straße, ein Restaurant oder einen Wald ergibt sich hier aus den verwendeten Tags und dem Typ des Elements. Daraus lässt sich auch der FCC bestimmen. Es war schon sehr früh klar, dass es sinnvoll ist GDF-Features dem Konverter schrittweise hinzufügen zu können um eine iterative Vorgehensweise zu ermöglichen. Die Idee war nur OSM-Objekte zu verarbeiten, für die ein FCC ermittelt werden kann. Der `FeatureClassResolver` entscheidet also darüber, welche Elemente in das GDF-Format exportiert werden. Anfangs wurde der FeatureClassCode noch mit einfachen `if/else`-Konstukten ermittelt, was sehr aufwendig und fehleranfällig war. Da dieser Code ständig erweitert werden musste, enstand irgendwann die Idee die ständigen Vergleiche durch eine geeignete Datenstruktur zu ersetzen. Abbildung 6.14 zeigt das grundsätzliche Prinzip der FCC-Ermittlung. Der `FeatureClassResolver` verwaltet eine Map in der, für jeden Schlüssel relevanter OSM-Tags, wieder eine Map enthalten ist. Diese enthält dann für den jeweiligen Wert des Schlüssels einen Array, mit je einem Element für jeden OSM-Typ. Darin ist dann der zugehörige FeatureClassCode enthalten.

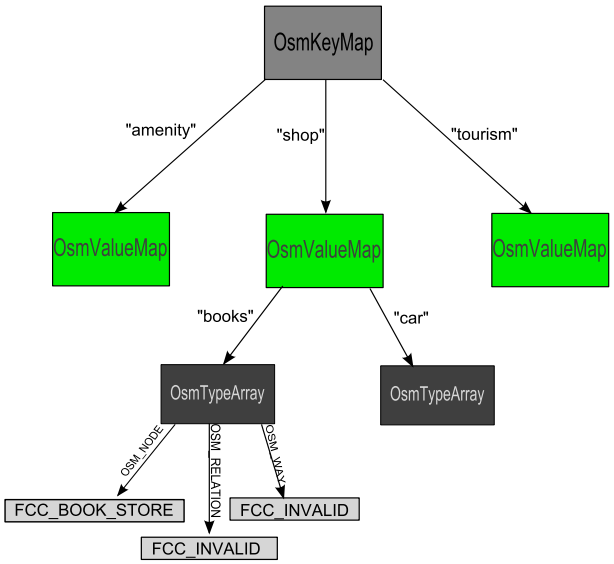


Abbildung 6.14.: Datenstruktur zur Ermittlung des FeatureClassCodes

Der Vorteil dieses Vorgehens zeigt sich in der einfachen Initialisierung der Struktur (Listing 6.4). Durch die einfache tabellarische Form sind Erweiterungen nun sehr leicht möglich. Zudem ist diese auch für eine Konfiguration über CSV- oder XML-Dateien gut geeignet. Ist die Struktur erst einmal initialisiert, wird für die Ermittlung des FCC kaum Rechenzeit benötigt.

```
AddTranslation("amenity","restaurant",OSM_NODE,FCC_RESTAURANT);
AddTranslation("amenity","fast_food",OSM_NODE,FCC_RESTAURANT);
AddTranslation("amenity","cafe",OSM_NODE,FCC_RESTAURANT);
AddTranslation("amenity","school",OSM_NODE,FCC_SCHOOL);
AddTranslation("amenity","bus_station",OSM_NODE,FCC_BUS_STATION);
AddTranslation("amenity","college",OSM_NODE,FCC_UNIVERSITY_OR_COLLEGE);
AddTranslation("amenity","university",OSM_NODE,FCC_UNIVERSITY_OR_COLLEGE);
AddTranslation("amenity","library",OSM_NODE,FCC_LIBRARY);
AddTranslation("amenity","place_of_worship",OSM_NODE,FCC_PLACE_OF_WORSHIP);
AddTranslation("amenity","fuel",OSM_NODE,FCC_PETROL_STATION);
```

Listing 6.4: Initialisierung des FccResolvers

Dem Klassendiagramm in Abbildung 6.15 ist zu entnehmen, dass es sich bei dem angesprochenen Array eigentlich um ein Klasse handelt, die dieses enthält. Dies ist sehr nützlich, denn der Konstruktor initialisiert bei der Erzeugung des Arrays dessen Felder automatisch zu `FCC_INVALID`. Der `FeatureClassResolver` selbst ist als Singleton[48] realisiert, damit die zeitintensive Initialisierung der Struktur nur einmal erfolgt. Die eigentliche Methode zur Ermittlung des FCC (`GetFeatureClass`) nimmt ein `OsmObject` und einen Array für die ermittelten Codes entgegen. Konnte mindestens ein FCC ermittelt werden, so wird `true` zurückgeliefert. Für ein einziges OSM-Element können mehrere FCCs zurückgeliefert werden. Ein einfaches Beispiel hierfür ist eine Brücke, die gleichzeitig auch eine Straße sein kann.

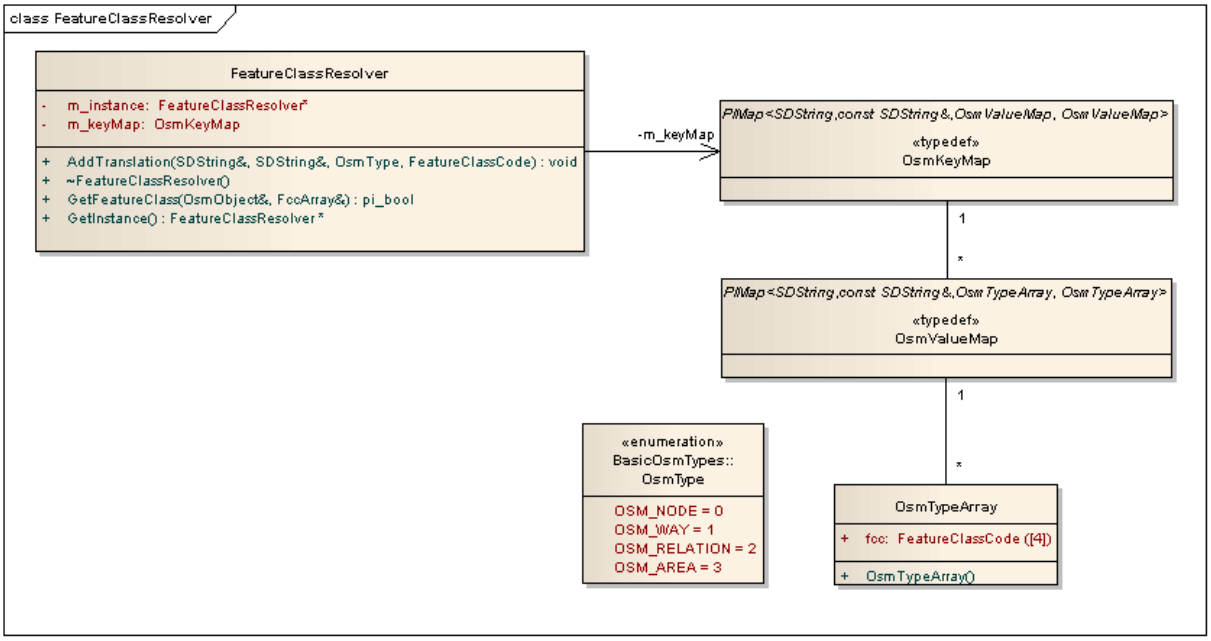


Abbildung 6.15.: Klassendiagramm FeatureClassResolver

[48] [Gamma, 2004, S. 157]

6.4.8. Erzeugung der GDF-Attribute

Attribute im GDF-Format werden durch einen Code in Form einer zweistelligen Zeichenkette angegeben. Der Wert eines Attributs ist entweder eine ganze Zahl, eine sehr kurze Zeichenkette mit maximal zehn Zeichen oder eine Referenz auf einen Eintrag in einer anderen Tabelle. Der Wert wird als Zeichenkette gespeichert und der verwendete Code entscheidet, als was diese interpretiert wird. Ebenso lassen sich Attribute in Form von sogenannten Segmenten gruppieren (Abbildung 6.16). Einige Attribute haben dabei nur eine sinnvolle Bedeutung, wenn sie sich gemeinsam in einem Segment befinden. Diese werden in der GDF-Referenz als **Composite Attributes** bezeichnet.

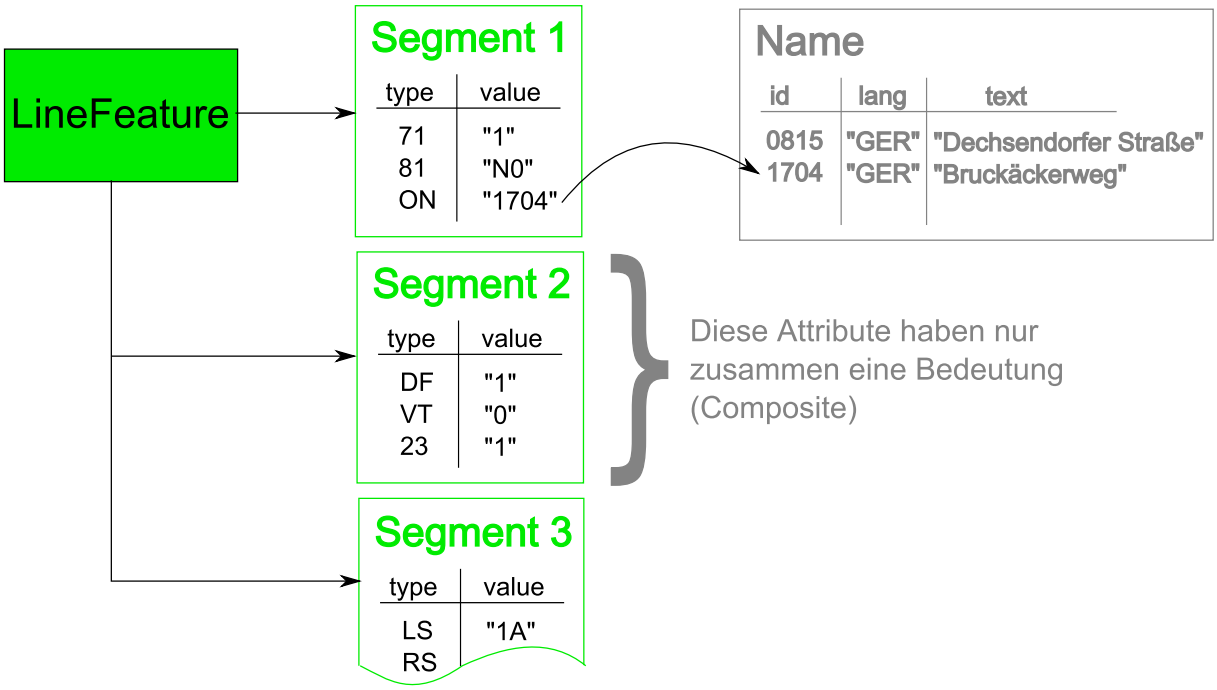


Abbildung 6.16.: Attribute im GDF-Format

Die Attribute sind in der GDF-Referenz nach **ThemeCodes** sortiert. Ein ThemeCode besteht dabei aus den ersten beiden Ziffern des vierstelligen FeatureClassCodes. Für den hier zu entwerfenden Konverter macht es also ebenfalls Sinn, die Ermittlung der Attribute nach ThemeCodes zu strukturieren. Hierdurch ist festgelegt, an welcher Stelle Änderungen vorzunehmen sind. Um dies zu erreichen wurde die Attributermittlung als Zuständigkeitskette[49] realisiert. Die in der Kette registrierten Klassen implementieren die abstrakte Methode `TranslateAttributes` der Basisklasse `AttributeTranslator` und kümmern sich um die Übersetzung von OSM-Tags in GDF-Attribute für einen je-

[49] [Gamma, 2004, S. 410]

weiligen ThemeCode (Abbildung 6.17). Gibt es sehr viele Attribute, die nur für einen bestimmten FeatureClassCode definiert sind, so kann auch eine Klasse hierfür geschrieben werden. Ermittelte Attribute werden mit Hilfe des `AttributeWriter` direkt in die Datenbank geschrieben. Mit Hilfe der Operationen `OpenGroup` und `CloseGroup` der Basisklasse kann ein neues Segment begonnen und wieder beendet werden. Dadurch lassen sich sehr leicht Segmente mit zusammenhängenden Attributen realisieren. Der `AttributeCollector` sammelt die IDs der so entstandenen Einträge und liefert diese als Ergebnis zurück. Damit die Initialisierung der Zuständigkeitskette nur einmal erfolgen muss, ist dieser als Singleton[50] realisiert. Listing 6.5 zeigt einen kurzen Ausschnitt einer möglichen Attributermittlung für eine Straße.

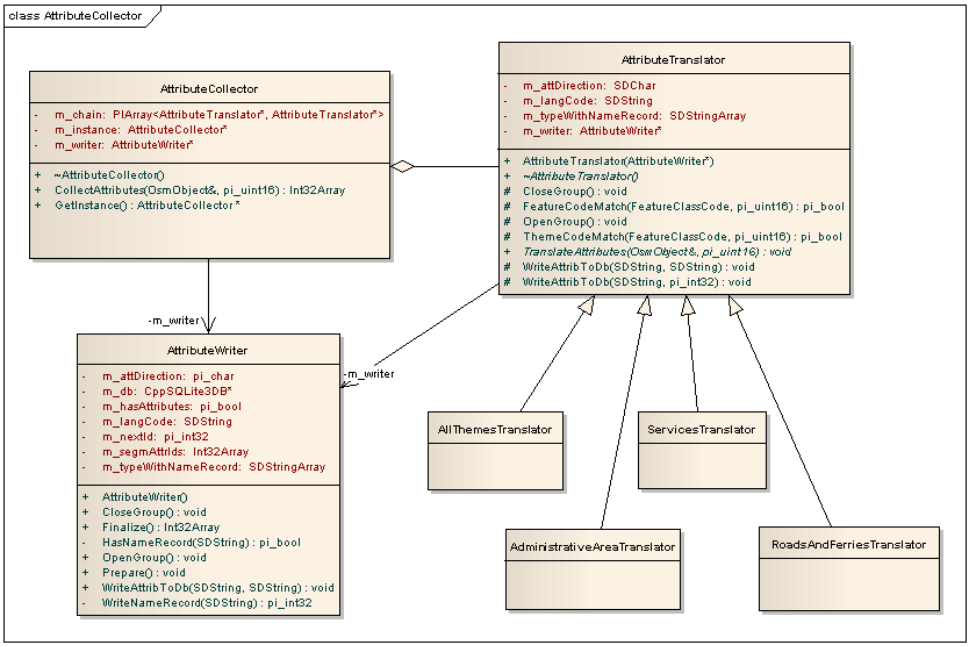


Abbildung 6.17.: Ermittlung der Attribute

```
1  void RoadsAndFerriesTranslator::TranslateAttributes(const OsmObject &obj, pi_uint16 fcc)
2  {
3      if ( !ThemeCodeMatch(TC_ROADS_AND_FERRIES, fcc) )
4          return;
5
6      WriteAttribToDb( ATTR_FORM_OF_WAY, GetFormOfWay(obj) );
7      WriteAttribToDb( ATTR_ROUTE_NUMBER, GetStreetReference(obj) );
8      WriteAttribToDb( ATTR_SPEED_CATEGORY, GetSpeedCategory(obj) );
9
10     // Composite Attribute: Access Characteristics
11
```

[50] [Gamma, 2004, S. 157]

```
12     OpenGroup();
13     WriteAttribToDb( ATTR_VEHICLE_TYPE, GetAllowedVehicleTypes(obj) );
14     WriteAttribToDb( ATTR_DIRECTION_OF_TRAFFIC_FLOW, GetDirection(obj) );
15     WriteAttribToDb( ATTR_THROUGH_TRAFFIC, IsTrafficAllowed(obj) );
16     CloseGroup();
17 }
```

Listing 6.5: Übersetzung der Attribute für eine Straße (Auszug)

6.4.9. Datenmodell für den Export der GDF-Elemente

Das Datenmodell für den Export der gewonnenen GDF-Daten, in Abbildung 6.18, sieht auf den ersten Blick etwas verwirrend aus. Im Grunde wird hier aber nur das GDF-Datenmodell nachgebildet. Lässt man die vielen Zuordnungstabellen weg, so erkennt man die Gemeinsamkeiten (Abbildung 6.19). ComplexFeatures werden bisher noch nicht auf diese Weise nachgebildet.

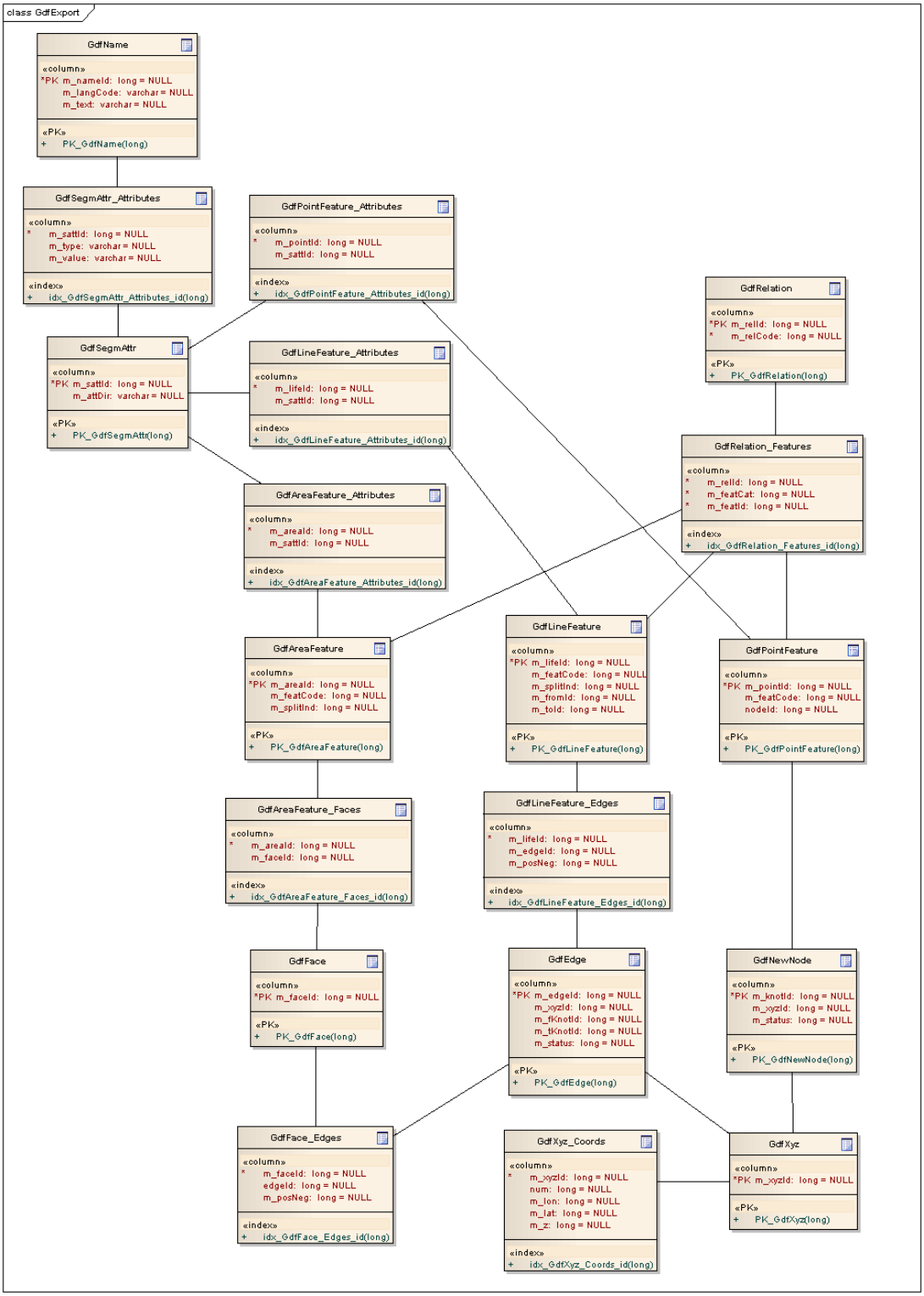


Abbildung 6.18.: Datenmodell für den GDF-Export

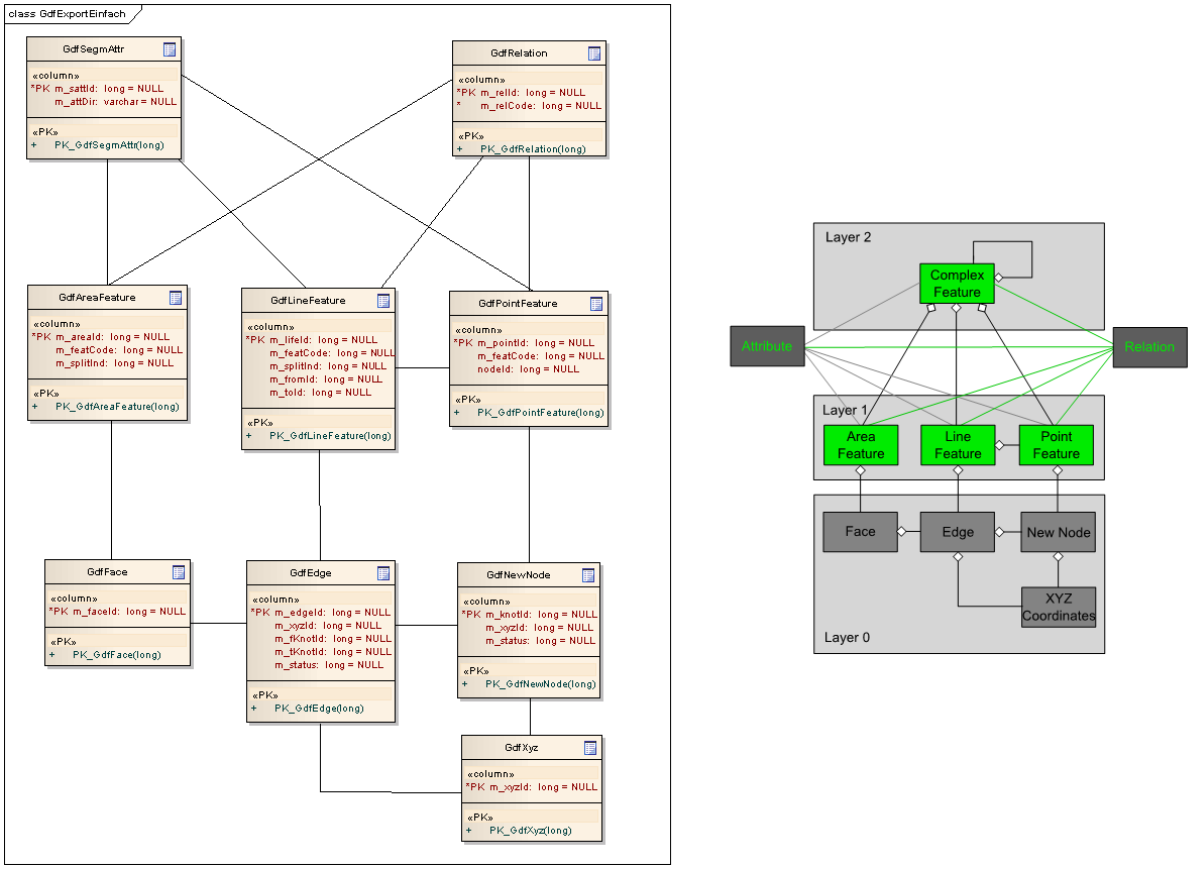


Abbildung 6.19.: Gemeinsamkeiten mit dem GDF Datenmodell

6.4.10. Export der grafischen GDF-Elemenete

Eine der wichtigsten Aufgaben des entwickelten Konverters ist die Realisierung einer funktionierenden Kartendarstellung. Zur Erklärung soll hier vorerst das in Abbildung 6.20 dargestellte, vereinfachte Datenmodell verwendet werden.

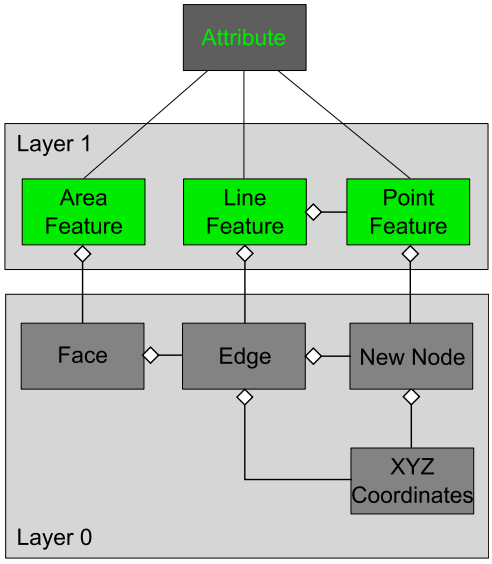


Abbildung 6.20.: vereinfachtes GDF Datenmodell

Auf unterster Ebene, die im GDF-Standard als Layer 0 bezeichnet wird, werden Geometrien beschrieben. Hier gibt es die sogenannten XYZ-Records. Diese können eine oder mehrere Koordinaten enthalten, wobei die Z-Dimension meist nicht verwendet wird und den Wert 0 enthält. Als Koordinatensystem wird hierbei, wie bei OpenStreetMap, WGS 84 verwendet. Punkte werden durch das Element NewNode dargestellt, welches als Position einen XYZ-Record mit nur einer Koordinate referenziert. Für Linien wird das Element Edge verwendet. Eine Edge verweist auf zwei NewNodes für den Start- und Endpunkt und auf einen XYZ-Record mit mehreren Koordinaten, die den Verlauf der Edge beschreiben. Flächen werden letztendlich durch ein oder mehrere Elemente des Typs Face beschrieben. Ein Face besteht hierbei wiederum aus einer oder mehreren Edges, die hier die Begrenzungslinien von Außen- und Innenflächen beschreiben. Die Elemente Point-, Line- und AreaFeature, im darüberliegenden Layer 1, ordnen diesen Geometrien mit Hilfe eines FeatureClassCodes und Attributen eine sinnvolle Bedeutung zu. Ein LineFeature kann dabei optional als Start- und Endpunkt auf zwei PointFeatures verweisen. Abbildung 6.21 zeigt die Datenstrukturen, die aus den eben vorgestellten Grundelementen aufgebaut werden müssen. Attribute wurden, aus Gründen der Übersichtlichkeit, weggelassen.

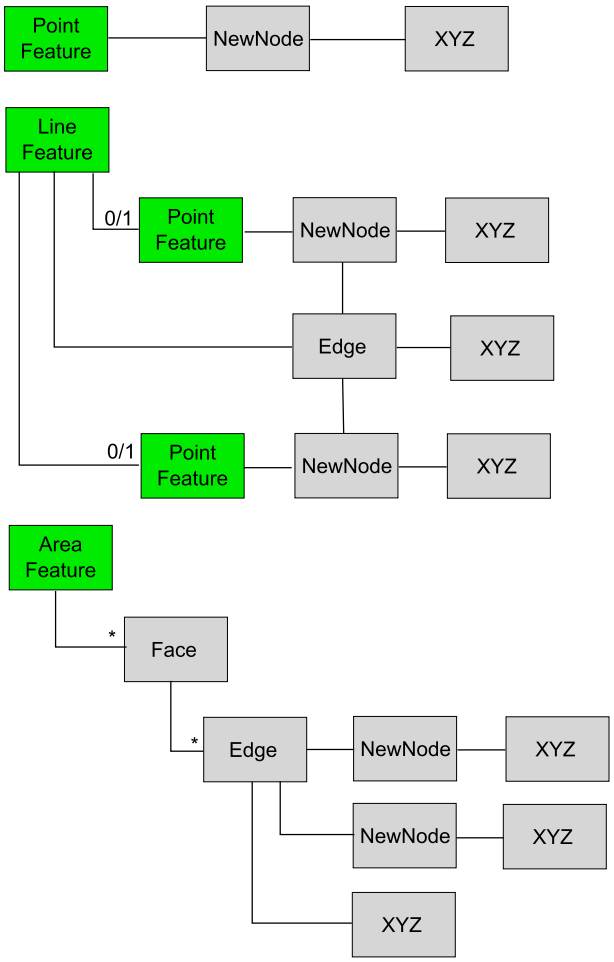


Abbildung 6.21.: Datenstrukturen aus den Grundelementen

Grundsätzlich stellt die Erzeugung dieser Strukturen kein Problem dar, denn die benötigten Informationen sind in OpenStreetMap alle vorhanden. Aus Nodes mit Tags lassen sich PointFeatures und aus Ways Line- oder AreaFeatures generieren. Auch die in OpenStreetMap als Multipolygon bezeichneten zusammengesetzten Flächen, die durch eine Relation beschrieben werden, lassen sich in das GDF-Format übertragen. Es stellt sich jedoch die Frage, wie sich diese möglichst einfach generieren lassen. Die in OpenStreetMap verwendeten IDs sind 64-Bit breit und es gibt keine sinnvolle Vorgehensweise, daraus die nur 32-Bit breiten IDs für das GDF-Format zu erzeugen. Es müssen also auf jeden Fall neue Kennungen vergeben werden. Zudem gibt es keinen wichtigen Grund auf einmal im GDF-Schema abgelegte Geometrien lesend oder verändernd zuzugreifen. Die Lösung, die hier gewählt wurde, basiert lediglich auf einigen elementaren Einfügeoperationen, die als Ergebnis eine neu generierte Kennung zurückliefern. Referenziert dabei ein Element auf Andere (wie beispielsweise eine NewNode auf einen XYZ-Record), so müssen diese vor diesem erzeugt werden

und als Parameter angegeben werden. Dies gewährleistet stets konsistente Daten. Durch geeignete Abfolgen der Einfügeoperationen, lassen sich alle in Abbildung 6.21 dargestellten Strukturen realisieren. Listing 6.6 zeigt die Verwendung am Beispiel eines einfachen LineFeatures.

```
void LineFeatureExport::AddSimpleLineFeature( const OsmWay &way, pi_uint16 fcc )
{
    AttributeCollector *attributeCollector = AttributeCollector::GetInstance();
    Int32Array attributeIds;

    attributeIds = attributeCollector->CollectAttributes(way, fcc);

    OsmNode fromNode = GetFromNode(way);
    OsmNode toNode = GetToNode(way);

    pi_int32 fromXyzId = InsertXyzRecord(fromNode);
    pi_int32 toXyzId = InsertXyzRecord(toNode);

    pi_int32 fromNodeId = InsertNewNodeRecord(fromXyzId);
    pi_int32 toNodeId = InsertNewNodeRecord(toXyzId);

    pi_int32 xyzGeometryId = InsertXyzRecord(way);

    pi_int32 edgeId = InsertEdge(xyzGeometryId, fromNodeId, toNodeId);
    pi_int32 lifeId = InsertLineFeature(fcc);

    AddEdgeToLineFeature(lifeId, edgeId);

    if (attributeIds.GetSize())
        AddLineFeatureAttributes(lifeId, attributeIds);
}
```

Listing 6.6: Export eines einfachen LineFeatures

In einem anderen Teil des Konverters werden räumliche Beziehungen zwischen einigen der exportierten Geometrien ausgewertet. Dazu müssen diese als OpenGIS-Datentypen in einem gesonderten Datenmodell abgelegt werden. Diese Aufgabe übernimmt ebenfalls der GdfExport, da hierfür die neu erzeugten IDs benötigt werden. Damit ist die Schnittstelle zur verwendeten Datenbank bereits festgelegt (Abbildung 6.22).

Abbildung 6.23 bietet eine Übersicht über das Modul GdfExport. Die Klasse `GeneralGdfExport` hat lediglich die Aufgabe, die Methoden der Datenbankschnittstelle ihren Unterklassen zur Verfügung zu stellen. Wie zu erwarten kümmern sich die Unterklas-

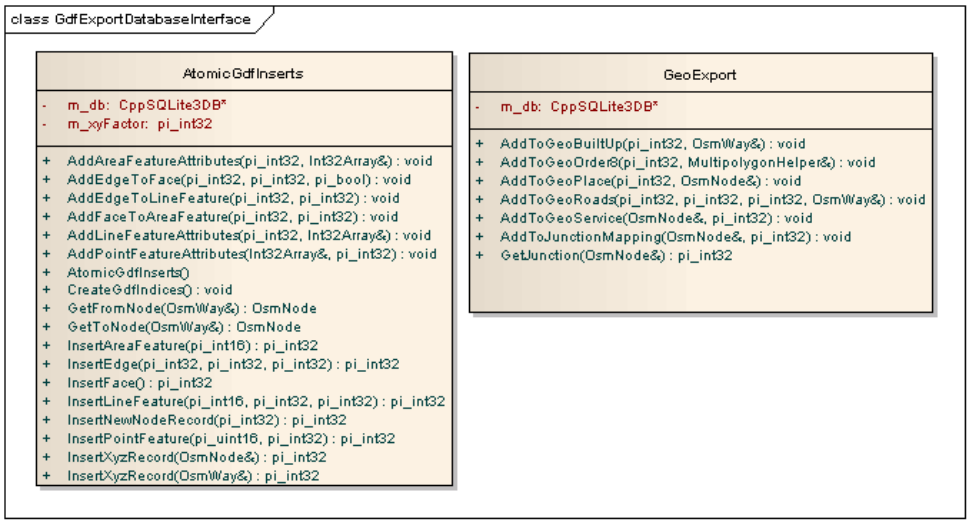


Abbildung 6.22.: Schnittstelle zur Datenbank

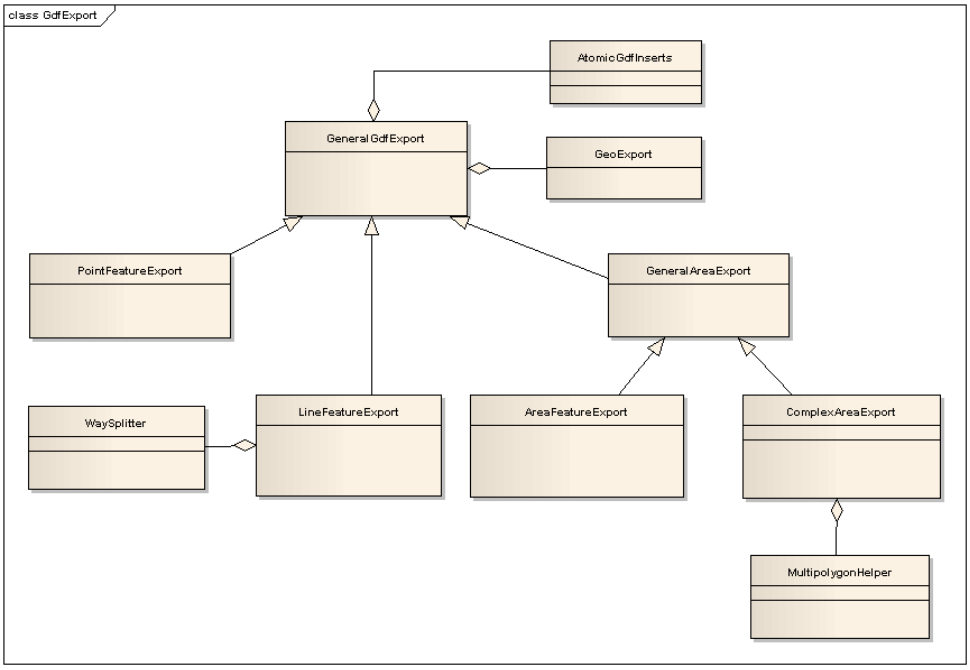


Abbildung 6.23.: Übersicht GdfExport

sen `Point-`, `Line-` und `AreaFeatureExport` um den Export der jeweiligen Elemente. `ComplexAreaExport` kümmert sich um die, in OpenStreetMap als Relation definierten, Multipolygone. Der Ablauf ist dabei fast immer gleich:

- Durchlaufen der Elemente eines OSM-Grundtyps (z. B. OsmWay)
- Ermittlung des FeatureClassCodes

- Ermittlung der Attribute aus FCC und Tags
- Abbilden der Daten in die GDF-Struktur
- Ggf. speichern der Geometrie im Geo-Modell für spätere Abfragen

Jedoch gibt es einige kleine Besonderheiten, die hier kurz erklärt werden.

PointFeatureExport

Wie zu erwarten werden hier aus OsmNodes PointFeatures erzeugt. Der Ablauf ist hierbei wie oben dargestellt. Allerdings werden Nodes, die als Kreuzung gekennzeichnet sind, in ein spezielles Mapping aufgenommen. Dadurch wird ein Zusammenhang zwischen einer Node in OSM und einem PointFeature hergestellt.

LineFeatureExport

Beim Export der LineFeatures wird diese Information wieder benötigt. Ways die Kreuzungen enthalten werden nämlich hier, mit Hilfe der Klasse `WaySplitter`, an diesen aufgetrennt (Abbildung 6.24) und als einzelne Segmente exportiert. Diese Segmente beginnen und enden jeweils an einem PointFeature, das im GDF-Format eine Kreuzung darstellt. Mit der Information im erzeugten Mapping können den Nodes an den Trennstellen PointFeatures zugeordnet werden. Damit Start- und Endpunkt einer Edge im GDF-Format korrekt dargestellt werden, muss hier noch eine Besonderheit beachtet werden [51]. Der Startpunkt muss stets eine kleinere Latitude [52] als der Endpunkt besitzen. Ist diese bei beiden identisch, so gilt das Selbe für die Longitude [53]. Erfüllt eines der Segmente diese Bedingung nicht, so wird dessen Richtung umgekehrt.

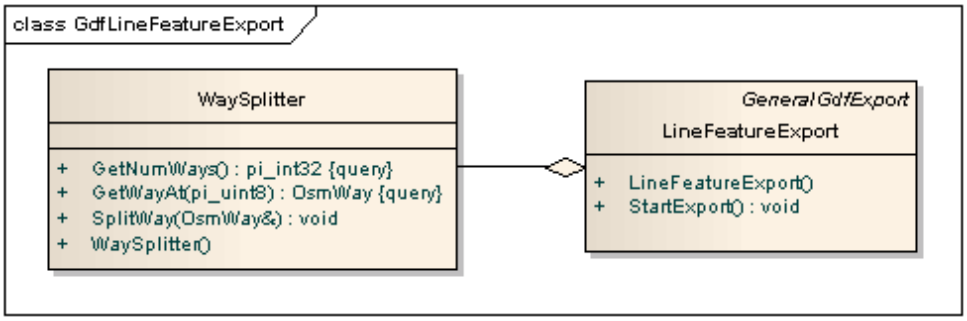


Abbildung 6.24.: LineFeatureExport

[51] [NAVTEQ, 2010, S. 2-6]
[52] X-Koordinate in WGS 84
[53] Y-Koordinate in WGS 84

AreaFeatureExport

Hier werden einfache Flächen, die nur aus einer Begrenzungslinie bestehen, exportiert. Dabei gibt es nur eine kleine Besonderheit. Die GDF-Spezifikation schreibt nämlich vor, dass Begrenzungslinien von Außenflächen im Uhrzeigersinn verlaufen[54] Ist dies nicht der Fall, so wird die Richtung des entsprechenden Ways einfach umgekehrt.

ComplexAreaExport

ComplexAreaExport kümmert sich um Multipolygone. Da diese in OpenStreetMap durch Relationen dargestellt werden, müssen hierfür einige Dinge beachtet werden.

- **Relation ist unvollständig**
  In einem Auszug aus den OSM-Daten ist die Bedingung, damit eine Relation darin vorkommt, dass lediglich ein Bestandteil dieser sich im ausgewählten Bereich befindet. Unvollständige Relationen sind daher keine Ausnahme, sondern eher die Regel.
- **Kein Hinweis auf den Typ der Fläche**
  Relationen vom Typ Multipolygon enthalten meist nur ein einziges Tag. Dieses gibt zwar an, dass es sich um die Beschreibung einer Fläche handelt, jedoch nicht ob es dabei beispielsweise um einen Wald oder ein Gebäude geht. Deshalb muss der FeatureClassCode anhand der Bestandteile der Relation ermittelt werden.
- **Multipolygon stellt keine gültige Geometrie dar**
  Relationen werden nach ihrer Erstellung nicht mehr von den OSM-Editoren geprüft. Oft ist dies auch gar nicht möglich denn, wie bereits erwähnt, sind diese oft nur teilweise im bearbeiteten Kartenausschnitt enthalten. Daher ist es leicht möglich eine Relation, durch Veränderung eines ihrer Bestandteile, zu beschädigen. Im Falle von Multipolygonen führt dies oft zu ungültigen Geometrien. Meist weden versehentlich Begrenzungslinien aufgetrennt oder Flächenelemente überschneiden sich.

Die hier gewählte Lösung toleriert nach Möglichkeit auftretende Fehler. Es wird jedoch versucht, mit aussagekräftigen Fehlermeldungen, auf Fehler in den OSM-Daten hinzuweisen, damit diese gezielt behoben werden können. Flächen, die Teil eines Multipolygons sind, können als soche gekennzeichnet werden um eine doppelte Behandlung durch `AreaFeatureExport` zu verhindern.

[54] [NAVTEQ, 2010, S. 2-13]

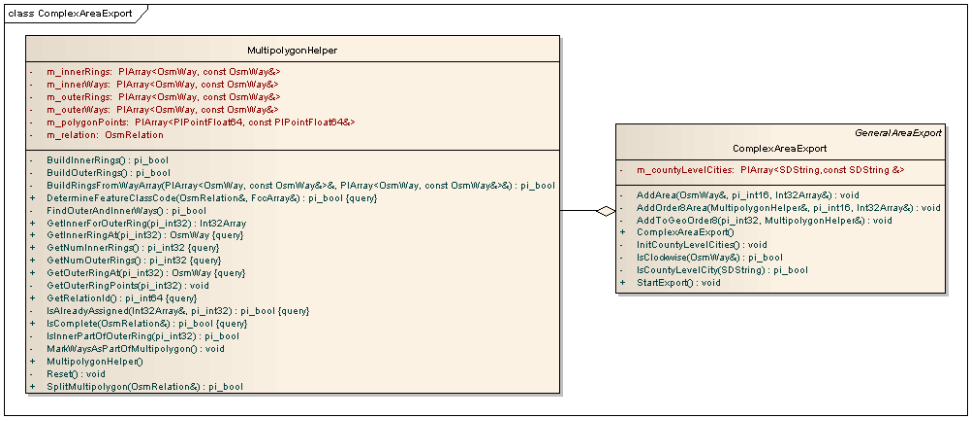


Abbildung 6.25.: ComplexAreaExport

Die Klasse `MultipolygonHelper` kümmert sich um die Besonderheiten im Zusammenhang mit Multipolygon-Relationen (Abbildung 6.25). Sie bietet Operationen zum Ermitteln der Vollständigkeit und des FeatureClassCodes. Aus den Ways, die die Begrenzungslinien der Flächenelemente beschreiben, werden Ringe gebildet. Dabei wird zwischen äußeren und inneren Ringen für Außen- und Innenflächen unterschieden. Die inneren Ringe (Innenflächen), die sich innerhalb eines Äußeren (Außenfläche) befinden, können abgefragt werden. Mit Hilfe dieser Informationen können leicht Face-Elemente oder OpenGIS-Geometrien erzeugt werden. Natürlich muss auch hier wieder der Uhrzeigersinn beachtet werden. Laut GDF-Spezifikation verlaufen die Begrenzungslinien für Außenflächen im und Innenflächen gegen den Uhrzeigersinn.

Beim Export der Geometriedaten für das im Anschluss beschriebene Geocoding wird hier noch eine Besonderheit berücksichtigt. Die Grenzen von Gemeinden und Städten werden in der Regel durch Order 8-Elemente sowohl in OpenStreetMap als auch im GDF-Format dargestellt. Jedoch gibt es in Deutschland den Sonderfall der kreisfreien Städte, die in OSM als Order 6 (Kreisgrenze) dargestellt werden. Diese haben jedoch für die Zieleingabe eine identische Bedeutung wie Order 8-Elemente und werden vom `ComplexAreaExport` entsprechend umgewandelt. Um dies zu ermöglichen wird im Moment noch der Name der Kreisgrenze mit einer Liste der 144 kreisfreien Städte verglichen. In Zukunft soll dies jedoch durch die Kennzeichnung mit einem entsprechenden Tag, in Absprache mit dem OSM-Projekt, ersetzt werden.

6.4.11. Erzeugung von Informationen für das Geocoding

Neben einer guten Kartendarstellung ist für eine Navigationslösung eine gut funktionierende Zieleingabe erforderlich. Die Bestimmung einer Position, anhand einer Adresse, wird als Geocoding bezeichnet und benötigt Informationen zum administrativen Aufbau eines Landes. Abbildung 6.26 zeigt, am Beispiel Deutschlands, welche Möglichkeiten das GDF-Format bietet um solche Informationen abzubilden. Hierfür kommen zwei, bis jetzt noch nicht verwendete, GDF-Elemente zum Einsatz:

- **Relationen**
  können zwischen zwei oder auch mehreren GDF-Elementen eine Beziehung herstellen, die durch einen Relations-Code festgelegt ist. Der Code legt auch die Reihenfolge und Art der beteiligten Elemente fest
- **Complex Features**
  sind in der Lage andere GDF-Elemente zu gruppieren und mit Attributen zu versehen. Dabei können sie auch wieder andere ComplexFeatures beinhalten. Ebenso können sie Teil einer GDF-Relation sein.

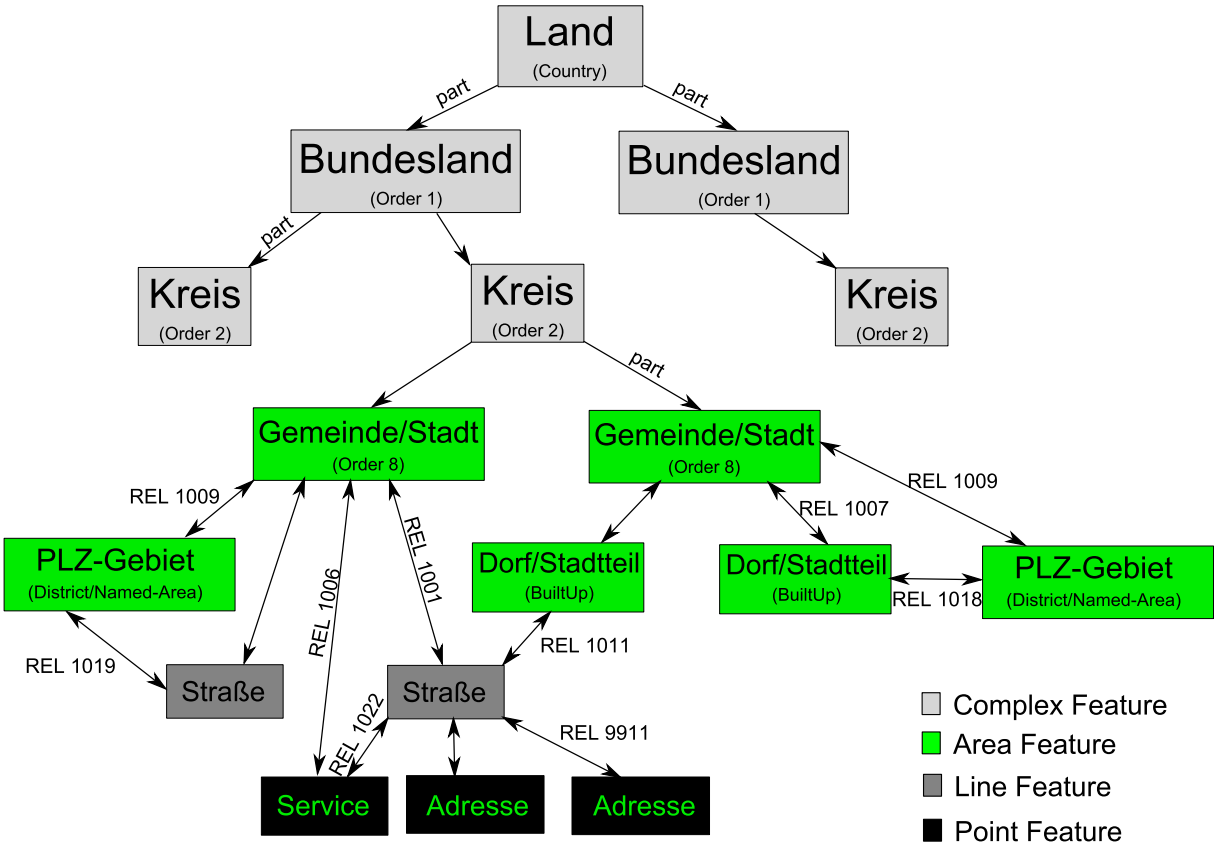


Abbildung 6.26.: Geocoding in GDF (Deutschland)

Die Struktur in Abbildung 6.26 kann jedoch von Land zu Land unterschiedlich aussehen. So hat beispielsweise ein kleines Land, wie Gibraltar, oft keine Entsprechungen für Order 1 oder Order 2-Elemente. Dies stellt jedoch kein großes Problem dar, denn die Zieleingabe beginnt in der Regel mit der Eingabe eines Stadt- oder Ortsnamens und somit auf der Ebene der Order 8-Elemente, die im GDF-Format flächendeckend vorhanden sein müssen. Gibt es dabei Mehrfachtreffer, so können diese anhand der Postleitzahlen unterschieden werden. Aus diesem Grund lässt sich die benötigte Information für das Geocoding stark vereinfachen. Abbildung 6.27 zeigt eine einfache Struktur, die bereits eine Zieleingabe mittels Orts- und Straßenname ermöglicht. Für Country und Order 1 wird hierbei jeweils ein Dummy-Element erzeugt. Dem Order 1-Element werden dann alle vorhandenen Städte und Gemeinden zugeordnet.

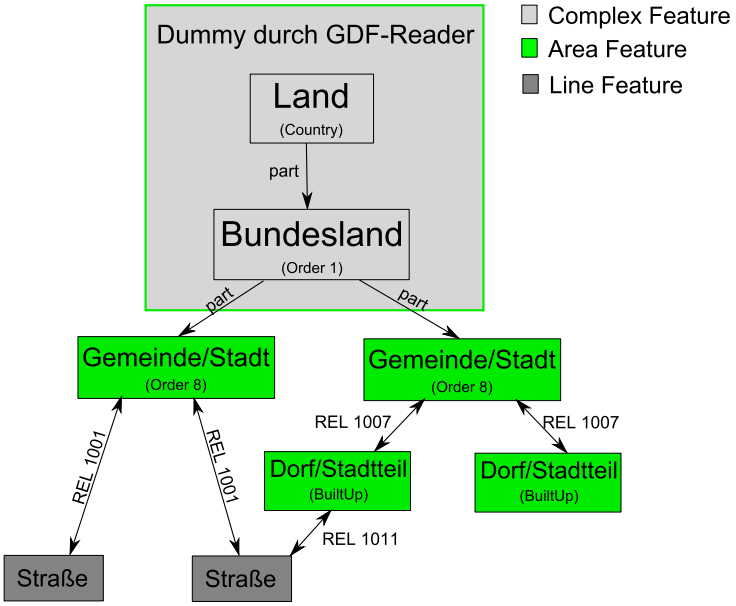


Abbildung 6.27.: vereinfachtes Geocoding

Nun müssen nur noch die Relationen erzeugt werden, welche die Beziehungen zwischen den verbleibenden Elementen ausdrücken. Tabelle 6.1 zeigt einen kleinen Ausschnitt der, zu diesem Zweck, im GDF-Format definierten Relationen.

Nachdem nun bekannt ist welche Daten im GDF-Format benötigt werden, stellt sich die Frage wie diese aus den OSM-Daten gewonnen werden können. Die benötigten Informationen werden beim Export der GDF-Elemente bereits im GeoModell abgelegt (Abbildung 6.28). Für eine Order 8-Area gibt es entsprechend in OpenStreetMap eine Multipolygon-Relation mit dem Tag `admin_level=8`, die grundsätzlich die selbe Bedeutung hat (GeoOrder8). Das zugehörige AreaFeature enthält normalerweise bereits alle nötigen Informationen zu Stadt-/Gemeindename und Postleitzahl. Für BuiltUps gibt es

code	name	involved features
1001	Road Element in Administrative Area	Order-8 Area, RoadElement
1006	Service in Administrative Area	Order-8 Area, Service
1007	BuiltUp Area in Administrative Area	Order-8 Area, BuiltUp Area
1011	Road Element in BuiltUp Area	BuiltUp Area, RoadElement
1016	Service in BuiltUp Area	Order-8 Area, Service
9911	Point Address along Road Element	Point Feature, Road Element

Tabelle 6.1.: Relationen für das Geocoding (Ausschnitt)

keine exakte Entsprechung. Jedoch sind Orte oder Stadtteile mit einer Fläche umgeben, die mit dem Tag `landuse=residential` gekennzeichnet ist (GeoBuiltUp). Diese enthält jedoch meist keine Information zum Namen des Ortes. Glücklicherweise befindet sich innerhalb einer solchen Fläche meist ein Punkt, der den Ortsmittelpunkt kennzeichnet und die benötigte Information enthält (GeoPlace). Derartige Flächen, ohne Ortsmittelpunkt, gehören, aller Wahrscheinlichkeit nach, zu einem solchen Punkt innerhalb der nächsten Umgebung. Dadurch lassen sich die fehlenden Attribute des zugehörigen AreaFeatures erzeugen. Ebenso wird die Geometrie von Straßen (GeoRoads) und die Position von POIs (GeoService) gespeichert. Da Hausnummern in OpenStreetMap eine Seltenheit darstellen, werden diese vorerst noch nicht berücksichtigt.

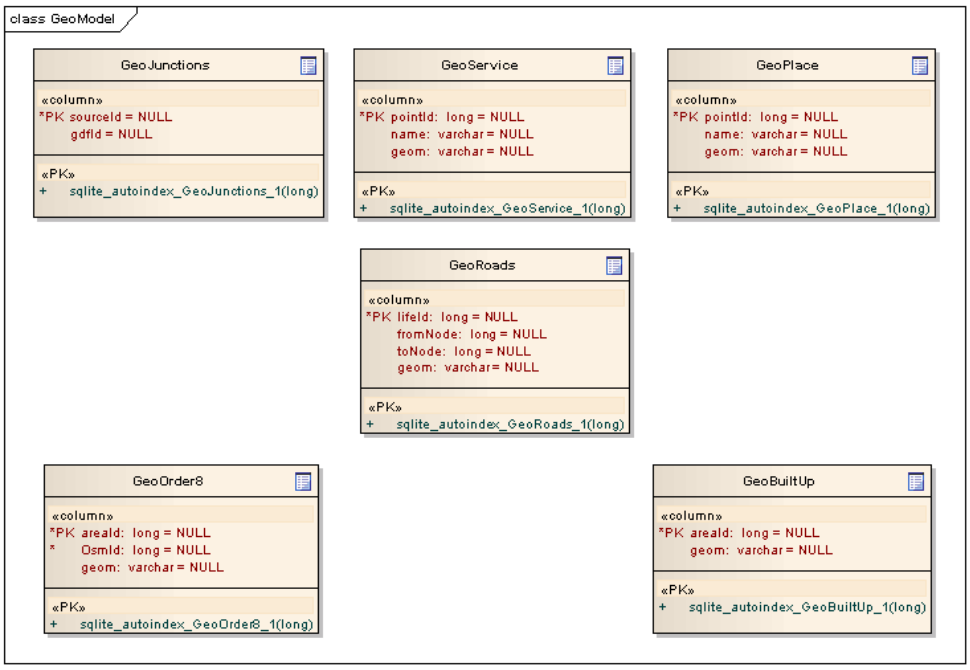


Abbildung 6.28.: GeoModel

Wie gesagt beginnt die Zieleingabe auf der Ebene von Städten und Gemeinden, die als Order 8 bezeichnet werden. Abbildung 6.29 zeigt die in GeoOrder8 gespeicherte Information für den Raum Mittelfranken.

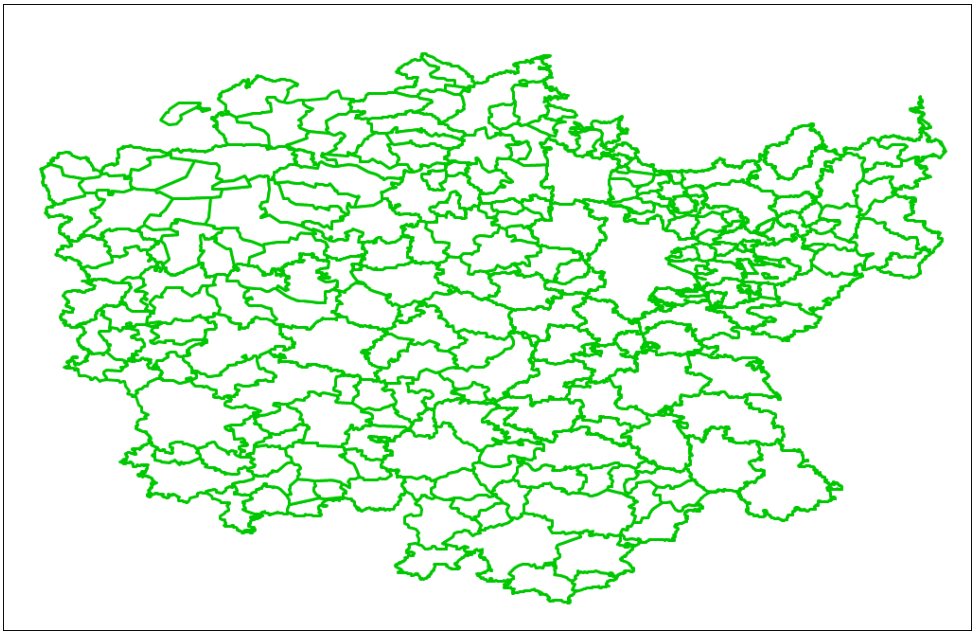


Abbildung 6.29.: Order8-Grenzen in Spatialite-GIS

In Abbildung 6.30 wurde zusätlich der Inhalt von GeoRoads eingeblendet. Die in einem Order 8-Element enthaltenen Straßen, lassen sich also leicht über eine geometrische Abfrage ermitteln. Die Information über enthaltene Sonderziele lässt sich auf gleiche Weise bestimmen.

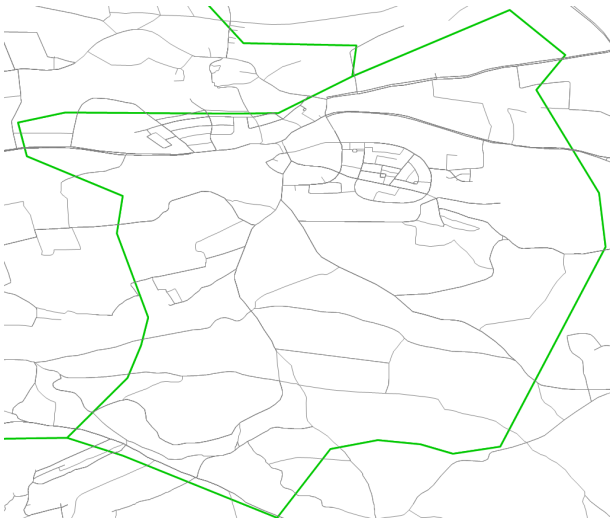


Abbildung 6.30.: Straßen innerhalb eines Order8

Abbildung 6.31 zeigt die Informationen aus GeoBuiltUp und GeoPlace an. Es ist nicht schwer nachzuvollziehen, wie der Name eines BuiltUps, mit Hilfe der Information in GeoPlace, ermittelt werden kann. Die Information über in einem BuiltUp enthaltene Straßen und Sonderziele lässt sich ähnlich einfach wie bei den Order 8-Elementen abfragen.

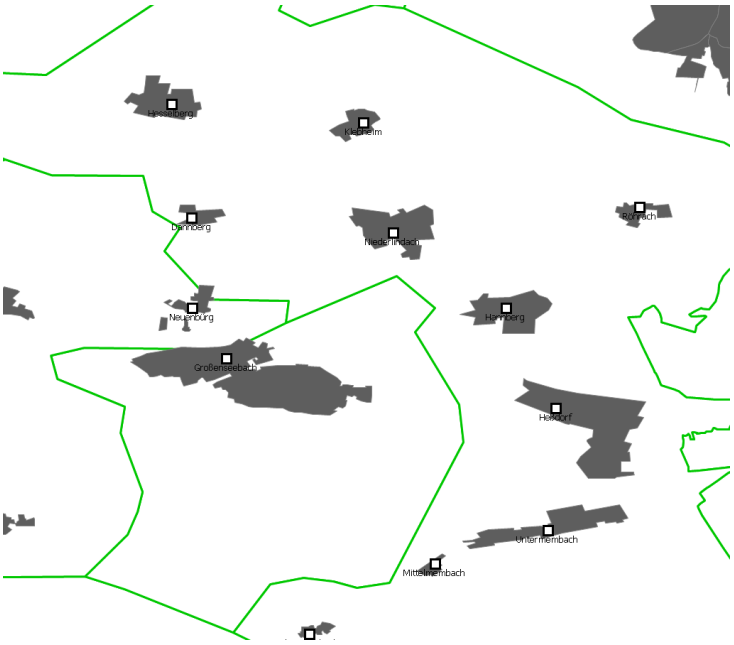


Abbildung 6.31.: BuiltUp und Place in Spatialite-GIS

Jedoch gibt es hier noch ein ungelöstes Problem. Wie am Beispiel von Großenseebach und Neuenbürg zu erkennen, gibt es Orte (BuiltUp) die sich nicht eindeutig innerhalb einer Gemeinde (Order8) befinden. Das Problem scheint auf dem ersten Blick sehr einfach, denn wahrscheinlich ist die richtige Gemeinde jene mit der größeren gemeinsamen Fläche. Doch was ist mit Straßen und Sonderzielen die sich geometrisch dem Ort zuordnen lassen, sich aber in den Grenzen einer anderen Gemeinde befinden. Um eine eindeutige Zuordnung zu gewährleisten, muss noch eine Lösung gefunden werden.

Die Klasse `GdfGeoRelationExport` kümmert sich als Erstes um die Beschaffung der fehlenden Informationen zu den BuiltUps. Die Methode `ResolveBuiltUpAreaName` erzeugt, mit Hilfe einer Abfrage der Ortsmittelpunkte (GeoPlace) innerhalb eines BuiltUp, die benötigten Attribute für das zugehörige AreaFeature. Mit Hilfe einiger weiterer geometrischer Abfragen werden die Relationen für das Geocoding erzeugt.

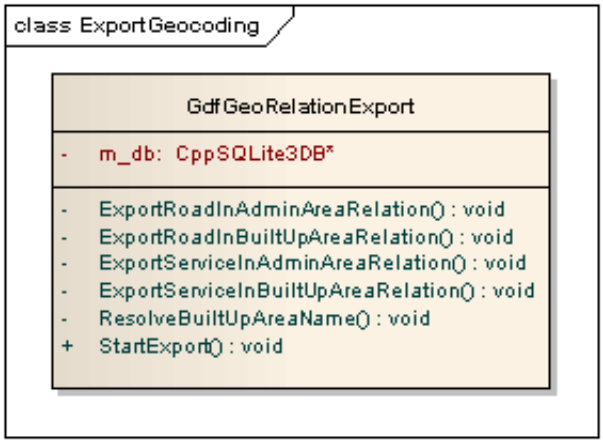


Abbildung 6.32.: ExportGeocoding

6.4.12. Realisierung einer einfachen Konfiguration

Für die Konfiguration des Konverters wird ein einfaches Konzept verwendet, das sich leicht um neue Attribute erweitern lässt. Als Vorbild hierfür diente die Klasse `QSettings` der Qt-Bibliothek [55]. Die Einstellungen werden in einer Map mit Schlüssel-Wert-Paaren vom Typ String verwaltet. Dazu gibt es einen XML-Parser, der die Dokumentstruktur in einer Art Dateipfad abbildet.

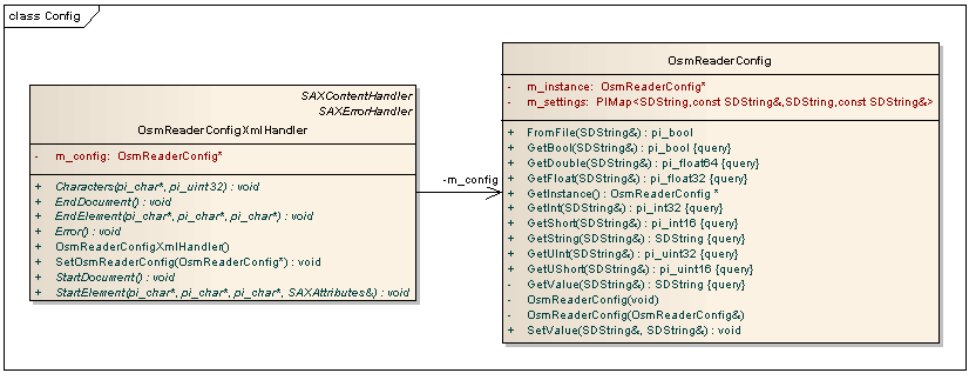


Abbildung 6.33.: Konfiguration

```
OsmReaderConfig *config = OsmReaderConfig::GetInstance();
config->FromFile(m_configFile);

pi_bool useLogfile = config->GetBool("logfile/use");
SDString logFile = config->GetString("logfile/value");
SDString dbFile = config->GetString("output_file/value");
SDString osmFile = config->GetString("input_file/value");
```

Listing 6.7: Verwendung der Konfiguration

6.5. Verwendete Algorithmen

Für einige in dieser Arbeit aufgetretenen Probleme waren Standardlösungen verfügbar. Dieser Abschnitt stellt einige, bei der Implementierung des Konverters verwendeten, Algorithmen vor.

[55] http://qt.nokia.com/

6.5.1. Berechnung des Uhrzeigersinns

Da die GDF-Spezifikation für Kanten von Flächen Uhrzeiger- bzw. Gegen-Uhrzeigersinn vorschreibt (Außenfläche im Uhrzeigersinn, Innenflächen dagegen) musste ein Algorithmus gefunden werden, der den Uhrzeigersinn von Polygonen ermittelt. Eine Möglichkeit, die dafür meist vorgeschlagen wird, ist die Verwendung des Vektorprodukts [56].

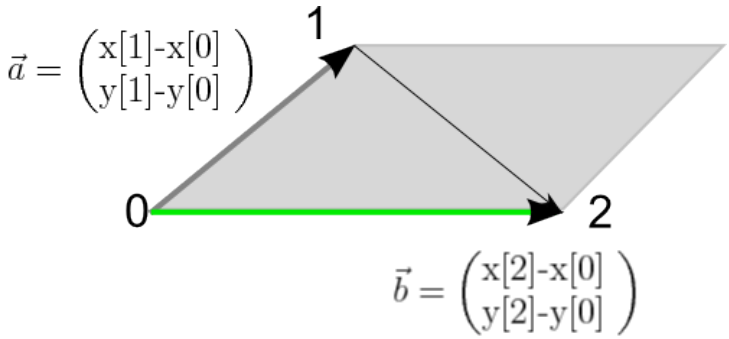


Abbildung 6.34.: Beispiel für zwei Vektoren im Uhrzeigersinn

Das Vektorprodukt liefert als Ergebnis einen Vektor, der senkrecht auf der von den Vektoren $\vec{a}$ und $\vec{b}$ aufgespannten Ebene steht. Sein Betrag entspricht der Fläche des, von den beiden Vektoren aufgespannten, Parallelogramms. Ist der Betrag der Fläche negativ, so beschreiben die beiden Vektoren einen Weg im Uhrzeigersinn. Das Vektorprodukt ist für Vektoren im dreidimensionalen Raum definiert. Die Formel für zweidimensionale Vektoren lässt sich jedoch leicht herleiten (Gleichung 6.1). Da das Ergebnis nur eine Dimension besitzt, lässt sich der Betrag einfach ablesen.

$$\begin{pmatrix} a_x \\ a_y \\ 0 \end{pmatrix} \times \begin{pmatrix} b_x \\ b_y \\ 0 \end{pmatrix} = \begin{vmatrix} \vec{e_x} & \vec{e_y} & \vec{e_z} \\ a_x & a_y & 0 \\ b_x & b_y & 0 \end{vmatrix} = (a_x b_y - a_y b_x)\vec{e_z} \tag{6.1}$$

$$= (x[1] - x[0]) \cdot (y[2] - y[0]) - (x[2] - x[0]) \cdot (y[1] - y[0]) \tag{6.2}$$

$$= x[1] \cdot y[2] - x[2] \cdot y[1] \tag{6.3}$$

Setzt man für die Vektoren die verwendeten Punkte ein, erhält man Gleichung 6.2. Wird für Punkt 0 der Koordinatenursprung verwendet, vereinfacht sich die Formel nochmals

[56] [Orwant, 2000, S. 441]

zu Gleichung 6.3. Leider ist die Ermittlung des Uhrzeigersinns mit drei Punkten nur bei konvexen Polygonen ausreichend. Eine einfache Möglichkeit der Verwendung für beliebige Polygone ist, an der Begrenzungslinie eines Polygons entlang zu laufen und die Fläche für jeweils zwei Punkte zu berechnen. Die Summe der berechneten Flächen gibt, anhand ihres Vorzeichens, den Uhrzeigersinn an.

```
pi_bool GeneralAreaFeatureExport::IsClockwise( const OsmWay &way )
{
    OsmNodeCursor nodeCursor;
    OsmNode node;
    pi_float64 x[2], y[2];
    pi_float64 area = 0.0;

    if (way.GetNodesCount() < 3)
        return false;

    nodeCursor.SelectWayNodes(way.GetId());
    node = nodeCursor.Next();
    x[0] = node.GetLat();
    y[0] = node.GetLon();

    while (nodeCursor.HasNext())
    {
        node = nodeCursor.Next();
        x[1] = node.GetLat();
        y[1] = node.GetLon();

        area += ((x[0] * y[1]) - (x[1] * y[0]));

        x[0] = x[1];
        y[0] = y[1];
    }

    if (area >= 0.0)
        return false;
    else
        return true;
}
```

Listing 6.8: Berechnung des Uhrzeigersinns

6.5.2. Punkt-In-Polygon

Für die Verarbeitung von Multipolygonen wird die Information benötigt, welche Polygone sich innerhalb der Grenzen eines anderen Polygons befinden. Hierfür wurde der Punkt-In-Polygon-Test nach Jordan verwendet.

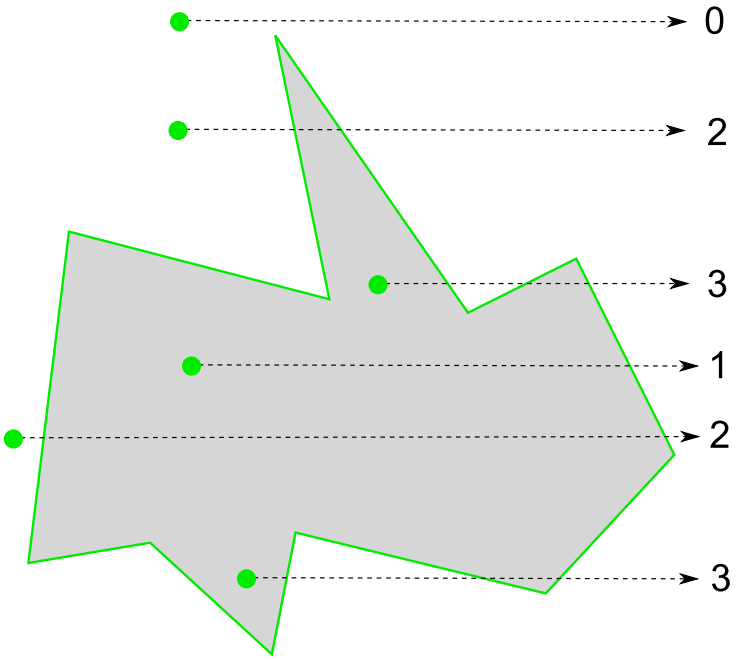


Abbildung 6.35.: Punkt-In-Polygon nach Jordan

Prinzipiell wird hier vom interessierenden Punkt ein Strahl ausgesandt und die Schnitte mit den Kanten des Polygons gezählt. Ergibt sich dabei eine ungerade Anzahl, so liegt der Punkt innerhalb. Zur Prüfung, ob der Strahl eine Kante des Polygons schneidet, wird hier wieder das Vektorprodukt verwendet. Dieses liefert nur im Falle eines Schnittes einen, von Null verschiedenen, Wert.

6.5.3. Multipolygon

Für die korrekte Verarbeitung von Multipolygonen gibt es einige, von der OpenStreetMap-Community, empfohlene Algorithmen [57]. Diese dienen dem Zweck der Ringbildung aus den beteiligten Way-Elementen, um geschlossene Linien für die Beschreibung einer Fläche zu finden und der Zuordnung von Innen- zu Außenflächen. Leider wurden diese erst entdeckt, nachdem eine funktionierende Lösung bereits implementiert war. Die

[57] [OpenStreetMap-Wiki, 2011, Relation:multipolygon/Algorithm]

Funktionsweise ist jedoch zu großen Teilen identisch. Eine Anpassung an den von OpenStreetMap empfohlenen Algorithmus is geplant.

6.6. Ergebnisse

Im Rahmen dieser Arbeit konnte ein Konverter für Daten aus dem OpenStreetMap-Projekt realisiert werden, der sich problemlos in den Kartenübersetzungsprozess der Firma Elektrobit integrieren lässt. Eine Erweiterung der Kartendarstellung ist über zwei, leicht zu erlernende, Erweiterungsschnittstellen möglich. Im Rahmen dieser Arbeit konnte dadurch eine, bereits sehr umfangreiche, Kartendarstellung realisiert werden (Abbildung 6.36).

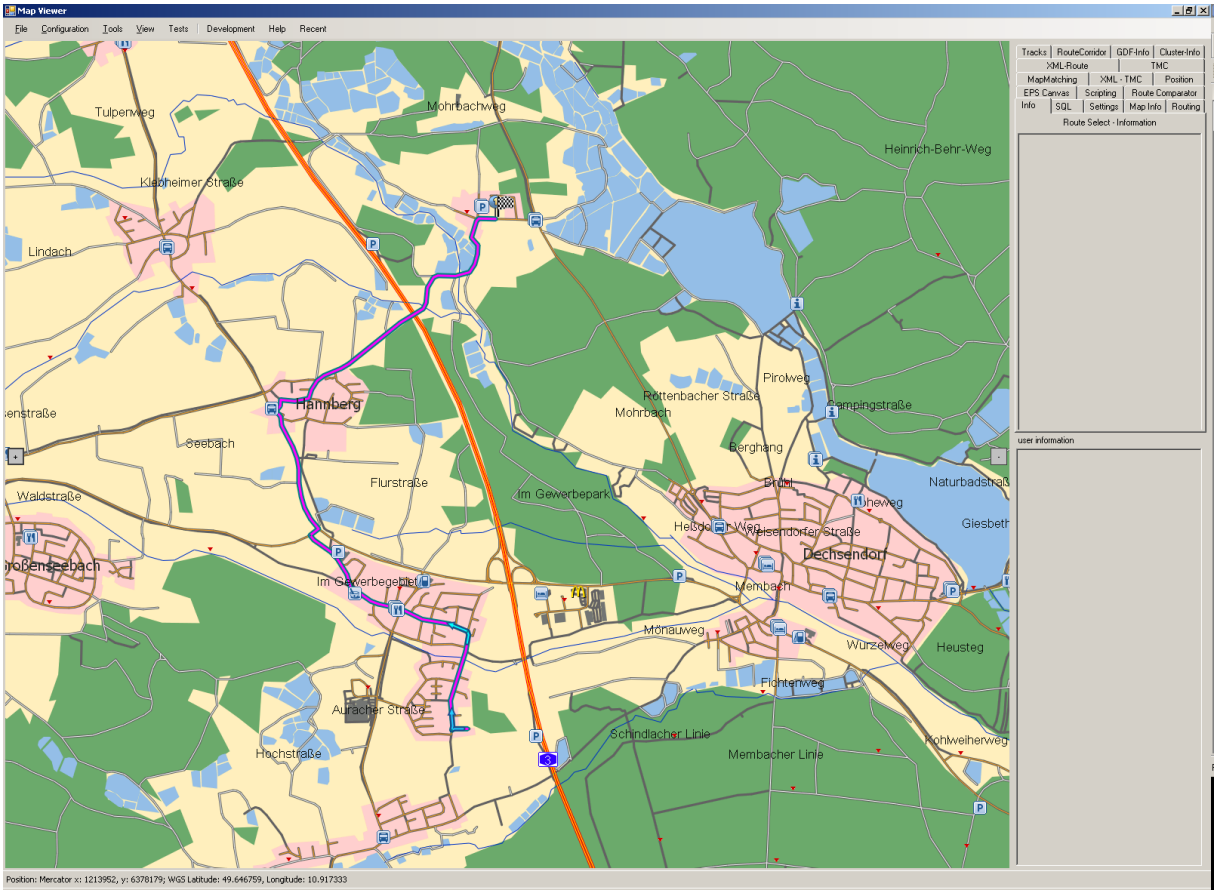


Abbildung 6.36.: Generierte Karte im MapViewer

Berücksichtigt wird dabei eine große Anzahl von unterschiedlichen Sonderzielen wie Restaurants und Hotels, Ärzte und Apotheken, Parkplätze und Tankstellen, Bushaltestellen, Schulen und Universitäten oder Ausflugsziele für Touristen. Zu den unterstützen Straßen zählen kleine Feld- und Radwege über Orts- und Kreisstraßen bis hin zu Kraftfahrstraßen und Autobahnen. Neben Gebäuden, Wäldern und Seen werden auch Ortsgrenzen oder Fußgängerzonen dargestellt.

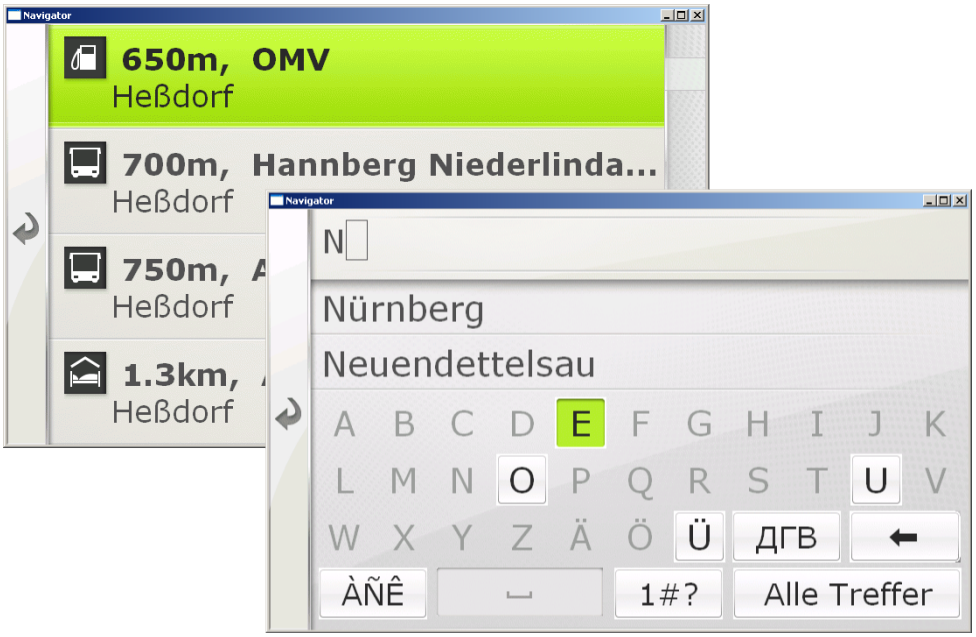


Abbildung 6.37.: Zieleingabe im EB street director

Die Zieleingabe berücksichtigt bereits die Eingabe eines Zielorts mit einer Straße oder einem Sonderziel. Sie kann für Deutschland und Länder mit einer vergleichbaren Verwaltungsstruktur, wie die Schweiz, Österreich, Spanien und vielen anderen, verwendet werden (Abbildung 6.37). Das gewählte Konzept verspricht dabei bessere Ergebnisse, als andere Lösungen auf OpenStreetMap-Basis.

Abbildung 6.38.: Zielführung im EB street director

Eine Routenberechnung mit anschließender Zielführung ist ebenfalls möglich (Abbildung 6.38). Aus den generierten Daten können präzise Fahranweisungen, mit Angabe von Straßennamen, gegeben werden. Zudem werden auch Informationen über spezielle Geschwindigkeitsbegrenzungen angezeigt. Anders als bei den Daten der kommerziellen Hersteller ist die Berechnung einer Route über Fuß-, Rad- oder Feldwege möglich. Trotz einiger noch fehlender Attribute für Straßen, sind die Ergebnisse der Routenberechnung bereits brauchbar.

Jedoch gibt es noch einiges zu tun. Es gibt noch eine große Anzahl von Features mit zugehörigen Attributen, die im Moment noch nicht berücksichtigt werden. Außerdem ist zu erwarten, dass für die Verarbeitung von Daten anderer Länder als Deutschland einige Anpassungen erforderlich sind. Es ist üblich, dass im PSF-Format eine Datei für jedes Land erzeugt wird. Dadurch sind Updates leichter möglich. Um dies zu ermöglichen ist eine der wichtigsten Aufgaben für die Zukunft, eine eindeutige Identifizierung und Kennzeichnung von Knoten und Kanten an den Landesgrenzen zu realisieren. Mit Hilfe dieser Information ist eine Routenberechnung und Zielführung auch über Landes- bzw. Dateigrenzen möglich. Zudem müssen noch einige spezielle Abbiegeverbote berücksichtigt werden, die in OSM als Relationen dargestellt werden.

7. Zusammenfassung und Ausblick

Die Firma Elektrobit (EB) bietet mit dem **EB street director** eine ausgereifte Navigationslösung, die gerne von Herstellern portabler Navigationsgeräte wie Falk oder Medion und für die OnBoard-Navigation vieler bekannter Automobilhersteller verwendet wird. Kommerzielle Anbieter von Navigationsdaten bieten ihre Karten meist in Form von sogenannten Geographic Data Files (GDF) an. Dies ist ein, auf der Norm **ISO 14825** basierendes, Textformat für vektorisierte Kartendaten, das sich bereits seit vielen Jahren in der Navigationsbranche etabliert hat. Aus diesen Daten wird bei Elektrobit ein firmeneigenes Format erzeugt, das speziell auf die Bedürfnisse von mobilen Navigationslösungen, mit geringen Hardware-Anforderungen, zugeschnitten ist. Mit dem OpenStreetMap-Projekt existiert nun schon seit einiger Zeit ein Angebot an, ebenfalls vektorbasierten, Kartendaten. Diese werden größtenteils von freiwilligen Helfern zusammengetragen und stehen kostenlos zur Verfügung. Neben der freien Verfügbarkeit weisen die Daten in OpenStreetMap (OSM) einige besondere Eigenschaften auf. So sind diese im Bereich von Städten und abseits von Straßen oft viel detaillierter als die Daten der kommerziellen Anbieter. Deren Angebot beschränkt sich meist auf Daten für die Navigation auf Straßen. Mit Hilfe der OpenStreetMap-Daten lassen sich daher auch Navigationslösungen realisieren, die für Fußgänger oder Fahrradfahrer geeignet sind.

Ziel dieser Arbeit war es ein Konzept zu erarbeiten, das es ermöglicht OpenStreetMap-Kartendaten im **EB street director** zu nutzen. Zu diesem Zweck wurde ein Konverter entwickelt, der das in **ISO 14825** beschriebene Datenmodell für Geographic Data Files in einer relationalen Datenbank nachbildet. Hierzu wurde Spatialite verwendet, eine Erweiterung der bekannten SQLite-Datenbank um, in OpenGIS beschriebene, geometrische Datentypen und Operationen.

Als das größte Problem bei der Umsetzung wurde die große Anzahl der, im GDF-Format beschriebenen, Kartenelemente und zugehöriger Attribute betrachtet. Aus diesem Grunde stand der Fokus bei der Entwicklung auf einer Lösung, die eine schrittweise Erweiterung der Kartendarstellung ermöglicht.

Mit Hilfe dieses Ansatzes konnte, im Rahmen dieser Arbeit, eine Lösung realisiert werden, die eine Erzeugung von Karten für den **EB street director** ermöglicht. Durch ein einfaches Konzept für das Hinzufügen neuer Kartenelemente, konnte eine bereits sehr umfangreiche Kartendarstellung erreicht werden. Die erzeugten Daten lassen sich dabei auch für Routenberechnung und Zielführung nutzen. Eine Zieleingabe, die eine Suche nach Ort- und Straßenname sowie nach Sonderzielen ermöglicht ist ebenfalls größtenteils realisiert.

Für die Zukunft gibt es jedoch noch einiges zu tun. So ermöglicht der **EB street director** die Karten einzelner Länder miteinander zu verbinden, um eine Navigation auch über Landesgrenzen hinweg zu ermöglichen. Um dies zu realisieren müssen die Kartenelemente, an den Grenzen eines Kartenausschnitts, speziell gekennzeichnet werden. Zudem werden spezielle Abbiegeverbote in den OpenStreetMap-Daten noch nicht berücksichtigt. Die Kartendarstellung lässt sich noch um viele Elemente erweitern und die Zieleingabe kann ebenfalls noch verbessert werden. Außerdem sind, besonders für die Zieleingabe, noch einige landesspezifische Anpassungen zu erwarten. Über weitere Attribute für Straßen, kann die Qualität der Routenberechnung noch verbessert werden.

Insgesamt ist zu bemerken, dass die Entwicklung eines vollständigen Konverters durchaus, mit vertretbarem Aufwand, realisierbar ist. Eine Eigenentwicklung bei EB bietet den Vorteil, dass spezielle Anforderungen bei der Kartenübersetzung und für die Verwendung im **EB street director** berücksichtigt werden können. Durch ein schrittweises Vorgehen bei der Erweiterung der Kartendarstellung um neue Elemente und Attribute, die ausreichende Tests vorsieht, kann eine gute Qualität der generierten Daten gewährleistet werden. Durch eine Konfiguration der, bei der Konvertierung berücksichtigten, Kartenelemente über CSV- oder XML-Dateien können auch spezielle Kundenwünsche berücksichtigt werden. Ein Einsatzgebiet, für die erzeugten Daten, ist beispielsweise die Realisierung einer Kartendarstellung für Gebiete, in denen sonst nur Straßendaten existieren. Dadurch kann dort die Orientierung erleichtert werden. Navigationslösungen für Fußgänger oder Fahrradfahrer sind ebenso denkbar, wie die Verwendung als Karte in Outdoor-Geräten. Obwohl einige Testfahrten mit dem Auto, trotz der wenigen bisher berücksichtigten Attribute für Straßen, bereits sehr gut verliefen, sollen die Daten hierfür vorerst nicht verwendet werden.

Es bleibt zu erwarten, dass sich das OpenStreetMap-Projekt ebenso rasant weiterentwickelt wie bisher und dadurch immer neue Anwendungsbereiche erschlossen werden können. Durch eine Verbesserung der bei OSM für das Erfassen der Daten verwendeten Tools um Funktionalitäten, die Aspekte wie etwa Routenberechnung oder eine Zieleingabe berücksichtigen, könnte schon bald eine kostengünstige Alternative zu kommerziellen Navigationsdaten entstehen.

Literaturverzeichnis

[Gamma 2004] Gamma: *Entwurfsmuster*. 2004 48, 49, 50

[Martin 2009] Martin, Robert C.: *Clean Code*. mitp, 2009 42

[NAVTEQ 2010] NAVTEQ: *NAVTEQ GDF 3.0 Reference Manual*. July 2010 27, 51, 54

[OGC 1999] OGC: *OpenGIS Simple Features Specification For SQL*. Version: May 1999. `http://www.opengeospatial.org/standards/sfs` 30

[OpenStreetMap-Wiki 2011] OpenStreetMap-Wiki: *OpenStreetMap-Wiki*. Version: Oct 2011. `wiki.openstreetmap.org`, Abruf: 22.10.2011 57

[Orwant 2000] Orwant, Jon: *Algorithmen in Perl*. O'Reilly, 2000 56

[Owens 2006] Owens, Michael: *The definitive Guide to SQLite*. Apress, 2006 5.3

[Streit 2011] Streit, Prof. Dr. U.: *Einführung in die Geoinformatik*. Version: Oct 2011. `http://ifgivor.uni-muenster.de/vorlesungen/Geoinformatik/index.html`, Abruf: 22.10.2011 5.2

[Topf 2010] Topf, Ramm: *OpenStreetMap*. Lehmanns Media, 2010 3

[Wikipedia 2011] Wikipedia: *Wikipedia*. Version: Sept 2011. `www.wikipedia.org` 31

[Zielstra 2010] Zielstra, Zipf: *Quantitative Studies on the Data Quality of OpenStreetMap in Germany*. Version: July 2010. `http://www.giscience2010.org/pdfs/paper_187.pdf` 2.6

A. Anhang

A.1. Verwendete Tools

A.1.1. Entwicklung

- **Microsoft Visual Studio 2008**
  Für die Programmierung in C++ wurde die Entwicklungsumgebung Visual Studio 2008 verwendet. Arbeitsbereiche wurden allerdings über CMake-Generatoren erzeugt.
  Website: `http://www.microsoft.com/germany/visualstudio`
- **Visual Assist X**
  Visual Assist ist ein Plugin für das Visual Studio, das neben umfangreichen Suchfunktionen auch einige einfache Refactorings bietet.
  Website: `http://www.wholetomato.com/`
- **Enterprise Architect**
  Der Enterprise Architect bietet Werkzeuge für alle Phasen eines Softwareentwicklungsprozesses. Neben einer großen Anzahl von unterstützten Diagrammtypen, kann aus UML-Diagrammen Code für verschiedene Programmiersprachen generiert werden oder aber Diagramme aus bereits bestehenden Code erzeugt werden.
  Website: `http://www.sparxsystems.de`
- **CMake**
  CMake ermöglicht es, in einer einfachen Skriptsprache, einen Übersetzungsprozess Plattform- und Compilerunabhängig zu definieren. Daraus können dann Makefiles oder Projekte für verschiedene Entwicklungsumgebungen generiert werden.
  Website: `http://www.cmake.org`
- **TortoiseSVN**
  Für die Versionsverwaltung des Codes wurde SVN eingesetzt. TortoiseSVN ermöglicht ein komfortables Arbeiten mit versionierten Dateien im Windows-Explorer. Dessen Kontextmenü wird dabei um einige SVN-Operationen erweitert und der Status der Dateien wird mit Hilfe von Overlay-Icons angezeigt.
  Website: `http://tortoisesvn.tigris.org`
- **Spatialite-GUI**
  Spatialite-GUI ist eine einfache grafische Oberfläche für die Spatialite-Datenbank.

Die Ergebnisse einer Abfrage werden als Tabelle angezeigt. Spalten die dabei Geometriedaten in Form eines BLOB's enhalten, können inspiziert werden.
Website: `http://www.gaia-gis.it/spatialite`

- **Spatialite-GIS**
  Spatialite-GIS realisiert ein einfaches Geoinformationssystem auf Basis der Spatialite-Datenbank. In der Datenbank gespeicherte Geometriedaten können durch Spatialite-GIS grafisch dargestellt werden.
  Website: `http://www.gaia-gis.it/spatialite`

- **Very Sleepy**
  Very Sleepy ist ein einfach zu benutzender Profiler für Windows. Mit Hilfe dieses Tools können sehr leicht die Gründe für eine lange Verarbeitungsdauer eines Codeabschnitts ermittelt und beseitigt werden. Anders als bei den meisten Profilern benötigt man aber für die Verwendung von Very Sleepy kaum Einarbeitungszeit. Zudem ist das Tool OpenSource.
  Website: `http://www.codersnotes.com/sleepy`

- **Doxygen**
  Für die Dokumentation des Quellcodes wurde Doxygen verwendet. Kommentare wurden allerdings sehr sparsam eingesetzt.
  Website: `http://www.doxygen.org`

- **PCLint**
  Für die statische Codeanalyse wurde PCLint eingesetzt. Durch die Überprüfung des Codes auf typische Programmierfehler, können Gründe für Laufzeitfehler verringert werden.
  Website: `http://www.gimpel.com`

- **Subsql**
  Subsql ist ein Bestandteil der objektorientierten Datenbank GigaBase. Mit Subsql können die in einer GigaBase-Datenbank vorhandenen Daten in einer tabellarischen Form betrachtet werden.
  Website: `http://www.garret.ru/gigabase.html`

A.1.2. OpenStreetMap

- **JOSM - Editor**
  JOSM ist ein, in der Programmiersprache Java programmierter, Editor für OpenStreetMap-

Daten. Er ermöglicht das Bearbeiten der OSM-Vektordaten und bietet eine Vielzahl an nützlichen Werkzeugen und Vorlagen für häufig benötigte Elemente. Zusätzlich können Informationen aus vielen verschiedenen Quellen eingeblendet werden.
Website: `josm.openstreetmap.de`

- **OSM Relation Analyzer**
  Mit dem OSM Relation Analyzer können Relationen in den OSM-Daten überprüft werden. Hierzu existiert eine einfache Weboberfläche, in der die Kennung einer Relation eingegeben werden kann. Der Analyzer versucht aus den Bestandteilen einer Relation einen Graphen aufzubauen. Ist das nicht möglich, so wird ein Fehler gemeldet.
  Website: `http://ra.osmsurround.org`

A.1.3. Elektrobit

- **MapViewer**
  Mit dem MapViewer kann der Inhalt von PSF-Dateien betrachtet werden. Neben der Kartendarstellung können auch Funktionen wie Zieleingabe oder Routenberechnung getestet werden. Besonders hilfreich ist die Möglichkeit Informationen in den, bei der Erzeugung verwendeten, Datenbanken anhand einer Auswahl von Kartenelementen abfragen zu können.

- **EB street director Application**
  Die PC-Portierung der EB street director Application ermöglicht ebenfalls eine Überprüfung von Kartendarstellung, Zieleingabe oder Routenberechnung. Jedoch kann zusätzlich die Zielführung entlang einer berechneten Route simuliert werden. Mit der EB street director Application auf einem WindowsCE-Gerät wurden zusätzlich mehrere Testfahrten unternommen.

- **GDFReader**
  Der GDFReader ermöglicht das Einlesen von Kartendaten im GDF-Format in eine auf GigaBase-basierende Datenbank, die als Vorstufe für die Erzeugung des firmeneigenen PSF-Formats dient. Im Rahmen dieser Arbeit wurde der GDFReader um einen zusätzlichen Import erweitert.

- **PSFCreator**
  Der PSFCreator erzeugt aus der eben beschriebenen GDF-Datenbank das firmeneigene PSF-Format, das in Navigationsgeräten zum Einsatz kommt.

- **MapCompilerGUI**
  Die Werkzeuge für die Kartenübersetzung bei Elektrobit werden über XML-Dateinen konfiguriert. Die MapCompilerGUI realisiert eine einfach zu bedienende grafische Oberfläche zur Erzeugung der benötigten Konfigurationsdateien, die über XML-Schema-Dateien dynamisch konfiguriert wird.

A.1.4. Dokumentation

- **Inkscape**
  Inkskape ist das beste zur Zeit verfügbare Vektorgrafikprogramm für das SVG-Format. SVG basiert auf XML und entstammt einem W3C-Standard. Es kann bereits von den meisten Internetbrowsern dargestellt werden und ermöglicht auch Interaktivität und Animationen. In Zukunft wird SVG sicher noch eine größere Rolle spielen als bisher. Mit der OpenClipart-Bibliothek existiert bereits eine große Anzahl an freien SVG-Cliparts, die in Inkscape genutzt werden können.
  Websites: `http://inkscape.org`, `http://www.openclipart.org`
- **Gimp**
  Eines der besten freien Grafikprogramme für Pixelgrafiken wie Fotos oder Bitmaps. Gimp bietet dabei ähnliche Funktionalität, wie der kostenpflichtige Adobe Photoshop.
  Website: `http://www.gimp.org`
- **FreeMind**
  FreeMind ist ein schönes OpenSource-Tool zum Erstellen von Mind-Maps. Elemente können dabei auch HTML-Code mit Links oder Grafiken enthalten. Es können auch Ordnerstrukturen oder der Aufbau von Webseiten in eine MindMap überführt werden. Für eine Veröffentlichung ist neben verschiedenen Bildformaten auch ein Export als Flash-Programm möglich, der eine interaktive Nutzung (ein- und ausklappen der Unterelemente eines Knotens) ermöglicht.
  Website: `http://freemind.sourceforge.net`
- **LaTeX**
  Diese Arbeit wurde mit LaTeX gesetzt. Als Distribution wurde MikTeX verwendet und als Editor das TeXnicCenter.
  Websites: `http://miktex.org`, `http://www.texniccenter.org`

A.1.5. Andere nützliche Tools

- **CamStudio**
  Mit dem OpenSource-Tool CamStudio kann ein Ausschnitt des Desktops als Video aufgezeichnet werden.
  Website: `http://camstudio.org/`
- **Violet**
  Violet ist ein praktischer kleiner UML-Editor, der sich nicht durch einen großen Funktionsumfang, sondern durch eine einfache Bedienung und eine schöne Diagrammdarstellung auszeichnet.
  Website: `http://sourceforge.net/projects/violet/`
- **MSYS**
  MSYS bietet unter Windows eine Unix-artige Shell mit vielen nützlichen Kommandozeilentools wie grep, sed, sort und mehr.
  Website: `http://www.mingw.org/wiki/MSYS`
- **Process Explorer**
  Verbesserter Windows-Taskmanager. Bietet zusätzlich Informationen über die Verwendung von Dateien, Handles, Ports und Registrierungsschlüsseln eines Prozesses.
  Website: `http://technet.microsoft.com/de-de/sysinternals`
- **Programmer's Notepad**
  Toller OpenSource Texteditor mit Syntax-Highlight für viele verschiedene Programmier- und Auszeichnungssprachen. Leistungsfähige Scripting-Schnittstelle mit Python.
  Website: `http://www.pnotepad.org`
- **Desktops**
  Bietet für Windows mehrere Arbeitsflächen, wie unter Linux üblich.
  Website: `http://technet.microsoft.com/de-de/sysinternals`
- **DropBox**
  Bietet, über eine Art Netzwerklaufwerk im Web, die Möglichkeit Daten auf mehreren Rechnern zu Synchronisieren.
  Website: `http://www.dropbox.com/`

A.2. Glossar

AreaFeature (GDF) Ein Element mit dem flächenförmige Kartenelemente beschrieben werden können.

Binary Large Object Möglichkeit um große binäre Objekte, wie Bild- oder Audiodateien in einer Datenbank ablegen zu können.

B-Tree Indexstruktur die Daten, sortiert nach Schlüsseln, in einer baumartigen Struktur speichert. Einfüge-, Such- und Löschoperationen können dadurch sehr schnell ausgeführt werden.

Callback (dt. Rückruffunktion) Ermöglich die Registrierung einer Funktion bei einem Softwaremodul, das diese dann beim Eintritt eines dafür bestimmten Ereignisses aufruft.

Changeset (OSM) Anfrage an den OpenStreetMap-Server in Form eines XML-Dokuments um mehrere Änderungen an OSM-Daten auf einmal zu übertragen.

ComplexFeature (GDF) Ein ComplexFeature kann die GDF-Elemente Point-, Line-, Area- und sogar wieder ComplexFeatures gruppieren und diesen gemeinsame Attribute zuweisen.

Composite Attribute (GDF) Eine Gruppe von Attributen, die nur zusammen eine sinnvolle Bedeutung haben.

Digital Terrain Model Höheninformationen, die die Form der Erdoberfläche beschreiben.

Document Object Model Schnittstelle für den Zugriff auf XML- oder HTML-Dateien, die es ermöglicht dynamisch deren Inhalt zu verändern.

FeatureClassCode (GDF) Ein FeatureClassCode (FCC) ordnet einem Grundelement im GDF-Format eine sinnvolle Bedeutung zu. Diese wird in Form eines vierstelligen Zahlencodes angegeben.

Geografisches Informationssystem datenverarbeitungsgestütztes Informationssystem zur Erfassung, Verwaltung, Analyse, Modellierung und Visualisierung von Geodaten mit dem Ziel der Gewinnung von Geoinformationen.

Geographic Data File textbasiertes Vektorformat zum Austausch von digitalen Navigationsdaten.

Global Positoning System globales satellitengestütztes System, das eine Positionsbestimmung im erdnahen Raum ermöglicht

LineFeature (GDF) Ein Element mit dem linienförmige Kartenelemente beschrieben werden können.

MapFeature (OSM) Eine Kombination aus OSM-Grundelement mit entsprechendem Tag, für die es eine Kartendarstellung gibt.

Mapper (OSM) Eine Person, die Daten zum OpenStreetMap-Projekt beiträgt.

Member (OSM) Ein Bestandteil einer Relation. Kann vom Typ Node, Way oder Relation sein.

Multipolygon (OSM) Ein Polygon, das aus mehreren Außenflächen bestehen kann. Diese können wiederum mehrere Aussparungen in Form von Innenflächen enthalten.

Node (OSM) Element in OpenStreetMap, das einen Punkt darstellt. Ein Node besitzt eine geografische Länge und Breite und kann Attribute in Form von Tags besitzen.

Opcode Eine Zahl, die die Nummer eines Maschinencodes für einen bestimmten Prozessor (oder auch einer virtuellen Maschine) angibt.

Open Geospatial Consortium Gremium das Standards für die Verarbeitung von Geoinformationen festlegt

OpenGIS Standard für die Interoperabilität von Geoinformationssystemen

OSM-Element Node, Way oder Relation

Physical Storage Format Kartenformat für die Navigationslösung EB street director von Elektrobit

Point Of Interest sind Orte, die für den Nutzer eines Navigationssystems eine Bedeutung haben können. Oft wird dort eine bestimmte Dienstleistung angeboten oder es handelt sich um eine Sehenswürdigkeit.

PointFeature (GDF) Ein Element mit dem punktförmige Kartenelemente beschrieben werden können.

Relation (GDF) Mit einer Relation können Beziehungen zwischen GDF-Elementen modelliert werden. Ein Relations-Code gibt die Art der Beziehung an. Als Teile von Realtionen kommen alle Point- Line- und AreaFeatures in Frage.

Relation (OSM) Element in OpenStreetMap, mit dem Zusammenhänge zwischen Nodes und oder Ways modelliert werden können. Beispielsweise können aus mehreren einfachen Ways komplexe Polygone zusammengesetzt werden.

Renderer (OSM) Ein Programm, das aus OSM-Vektordaten Bilddateien generiert. Bekannte Renderer hierfür sind Osmarender, Mapnik oder Kosmos.

R-Tree erweitert das Konzept des B-Tree auf mehrere Dimensionen. Mit Hilfe eines R-Trees können beispielsweise Vektordaten anhand ihrer geografischen Koordinaten abgelegt werden. Eine Abfrage über einen geografischen Ausschnitt, der in einem R-Tree enthaltenen Daten, ist sehr schnell.

Tag (OSM) Ein Attribut für eines der OSM-Grundelemente in Form eines Schlüssel-Wert-Paares.

Text To Speech Ein Mechanismus der Anweisungen in Textform, durch eine künstliche Erzeugung der menschlichen Sprechstimme, ausgeben kann.

Uniform Resource Identifier Zeichenfolge, die zur Indentifizierung einer Resource (Webseite, Datei) im Internet dient.

Way (OSM) Element in OpenStreetMap, das eine Linie darstellen kann. Hierzu besteht ein Way aus einer geordneten Liste von Referenzen auf Nodes, die dessen Form beschreiben. Sind die erste und letzte Node dieser Liste identisch, so kann ein Way auch als Fläche interpretiert werden.

Web Map Service Schnittstelle zum Abrufen von Karteninformationen über das Internet, die meist in Form von kleinen Bildausschnitten (Kartenkacheln) zu Verfügung stehen.

Well-Known Binary Binäre Entsprechung des Well-Known Text Formats.

Well-Known Text Textuelles Austauschformat für vektorielle Geometriedaten.

World Geodetic System 1984 einheitliche Grundlage für Positionsangaben auf der Erde und im erdnahen Weltraum (für GPS verwendet).