Institut für Verteilte und Parallele Systeme

Abteilung Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Studienarbeit Nr. 2312

Visualisierung von Kartenobjekten mit GeoTools

Andreas Paul

Studiengang:	Informatik
Prüfer:	Prof. Dr. Kurt Rothermel
Betreuer:	Pavel Skvorzov
begonnen am:	01.12.2010
beendet am:	02.06.2011
CR-Klassifikation:	C.2.4, H.3.5

Zusammenfassung

"Location-based-Services" (LBS) sind Dienste, die Positionsinformationen mobiler Geräte verarbeiten. Da Benutzer ihre Position preisgeben müssen, um einen Service nutzen zu können, werden verschiedene Algorithmen entwickelt, welche den Aufenthaltsort verschleiern um die Privatsphäre zu schützen. Gleichzeitig soll aber auch eine für den Service ausreichende Genauigkeit der Position geboten werden. Da jeder Benutzer für unterschiedliche Orte andere Aufenthaltswahrscheinlichkeiten hat, spielt auch die konkrete Umgebung eine Rolle, welche für die Verschleierung beachtet werden muss. In dieser Arbeit soll eine Anwendung entwickelt werden, welche Wahrscheinlichkeitsabschätzungen für Objekte (Features) einer Karte grafisch darstellt. Dazu wird die GeoTools-Bibliothek genauer betrachtet und deren Möglichkeiten analysiert. Basierend auf GeoTools wird eine Anwendung entwickelt, welche als Eingabe entsprechend der Verschleierungspräferenzen die Aufenthaltswahrscheinlichkeiten des Benutzers bekommt und diese grafisch darstellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Motivation

Mit der fortschreitenden Verbreitung von mobilen Endgeräten mit GPS-Empfänger, wie z.B. Smartphones, gewinnen auch Dienste wie z.B. Restaurant- oder Friendfinder, welche die Positionsinformationen der Clients nutzen, eine größere Bedeutung. Dabei möchte aber nicht jeder Benutzer, dass jeder Service dessen genaue Position erfährt, es muss also irgendeine Art von Verschleierung durchgeführt werden. Im Gegenzug darf aber durch die Positionsverschleierung die Dienstqualtität nicht zu stark eingeschränkt werden, da der Dienst sonst seine Aufgabe nicht mehr erfüllen kann. Eine einfache Randomisierung der Positionsinformationen unabhängig von der Karte ist jedoch nicht ausreichend, da jeder Benutzer des mobilen Gerätes sich in verschiedenen Gebieten mit unterschiedlicher Wahrscheinlichkeit aufhält. Hierfür gibt es mehrere Forschungsansätze für Algorithmen, welche eine Verschleierung von Positionsinformationen durchführen.
in dieser Arbeit soll eine Anwendung erstellt werden, welche Aufenthaltswahrscheinlichkeiten eines Benutzers in einer Karte darstellt, um Verschleierungsalgorithmen zu visualisieren und überprüfen.

1.2 Aufgabestellung

Die Hauptaufgabe dieser Arbeit ist es Geodaten zu lesen und abhängig von ihren gegebenen Wahrscheinlichkeitsabschätzungen zu visualisieren. Die Wahrscheinlichkeitsabschätzungen für verschiedene Kartenobjekte und Attribute werden als vordefinierte Eingabewerte angenommen. Dies beinhaltet die folgenden Arbeitsschritte:

- Untersuchung der GeoTools Bibliothek
- grafische Visualisierung einer Karte im Shapefile-Format

- Erstellen eines Modells für die Repräsentation von feature-abhängigen Wahrscheinlichkeiten
- grafische Visualisierung der feature-abhängigen Wahrscheinlichkeitsverteilung

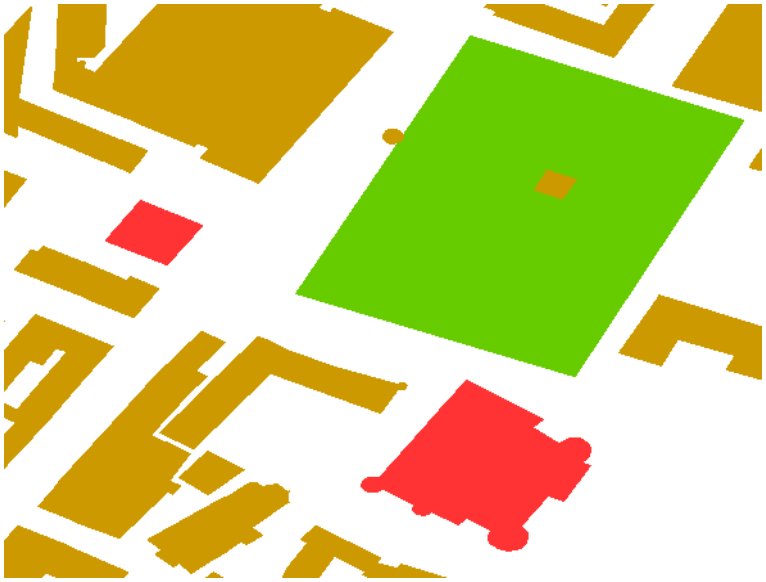


Abbildung 1.1: Beispiel für grafische Visualisierung von Features.

1.3 Szenarien

Die in dieser Arbeit erstellte Anwendung soll die Möglichkeit bieten, Karten bzw. deren Objekte anhand ihrer Eigenschaften unterschiedlich einzufärben. Dies kann entweder eine einfache Unterscheidung von Objektarten wie Gebäuden oder Straßen sein, oder auch anhand von komplexen Eigenschaften wie Aufenthaltswahrscheinlichkeiten von Personen in verschiedenen Gebäuden oder Plätzen. mit diesem Werkzeug ist z.B. eine grafische Darstellung von kartenbasierten Positionsverschleierungsalogrithmen möglich. Insbesondere sollen sich Aufenthaltswahrscheinlichkeiten für verschiedene Benutzertypen auf unterschiedlichen Karten visualisieren lassen.

2 Grundlagen und verwandte Arbeiten

2.1 Kartenanalyse für Positionsverschleierung

Protecting Privacy in Continuous Location-Tracking Applications

Die Autoren von [7] vergleichen drei einfache Algorithmen, mit denen die Anzahl sensitiver Positionsupdates, aus denen ein potenzieller Angreifer auf die Position schließen kann, verringert werden kann. Die Autoren gehen von einer Architektur, wie in Abbildung 2.1 dargestellt, aus. Der Benutzer bzw. dessen mobiles Endgerät (Data Subject) sendet regelmäßig seine aktuelle Position an einen Locationbroker. Dieser Locationbroker wird z.B. vom Netzbetreiber zur Verfügung gestellt zu dem der Benutzer aufgrund längerer Geschäftsbeziehungen größeres Vertrauen hat, als zu einem beliebigen anderen Dienst.

Der Locationbroker verwaltet die Datenschutzeinstellungen des Benutzers und leitet über den Notification Manager nur die Updates an den Service Provider weiter, die den Vorgaben des Benutzers entsprechen. Die Datenschutzeinstellungen eines Nutzers bestehen aus Angaben, wie stark er seine Position für verschiedene Gebiete verbergen will. Befindet sich der Benutzer in einem Gebiet, das er als unempfindlich einstuft leitet

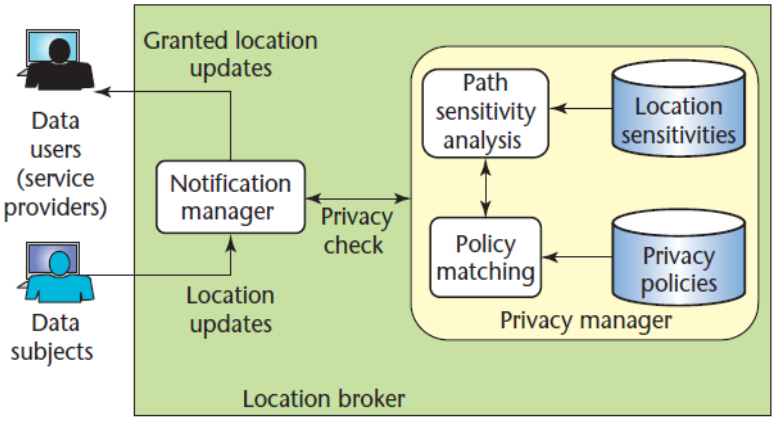


Abbildung 2.1: Architektur aus [7]

der Notification Manager alle Nachrichten an den Serviceprovider weiter um die Positionsgenaugkeit und damit die Servicequalität zu erhöhen. Interessant sind die sensitiven (empfindlichen) Gebiete. Durch verschiedene Algorithmen soll durch Unterdrückung von Positionsupdates die Unsicherheit eines "Angreifers" bezüglich der Position möglichst groß gehalten werden.

- Der **Base**-Algorithmus ist die einfachste Version. Hier werden nur Locationupdates an den Serviceprovider weitergeleitet, die außerhalb von sensitiven Gebieten erfolgen.
- Beim **Bounded-rate** Algorithmus wird zusätzlich darauf geachtet, dass die Updatefrequenz einen vordefinierten Schwellenwert nicht überschreitet. Wird der Schwellenwert niedrig gewählt, so ist es durch die größere zurückgelegte Strecke zwischen zwei Updates z.B. unwahrscheinlicher, dass ein Update direkt an einer Grenze zu einem sensitiven Gebiet passiert.
- Zuletzt gibt der **k-area**-Algorithmus Updates nur weiter, wenn nicht ersichtlich ist, welches der letzten k sensitiven Gebiete besucht wurde.

Zum Testen verwendeten die Autoren ein Stadtgebiet, in dem die Gebäude sensitive Gebiete und die Straßen unsensitive Gebiete sind. Es wurde ein einfacher automatisierter Angreifer simuliert, der bei fehlenden Updates die Position zwischen zwei Updates linear interpoliert und den Benutzer in dem sensitiven Gebiet vermutet, das am nächsten zur Mitte der Interpolationslinie ist. Der Angreifer hat sonst keine Informationen über die überwachten Objekte. In Grafik 2.2 werden die vom Angreifer erkannten Gebiete den veröffentlichten Locationupdates in unsensitiven Gebieten gegenübergestellt. Beim

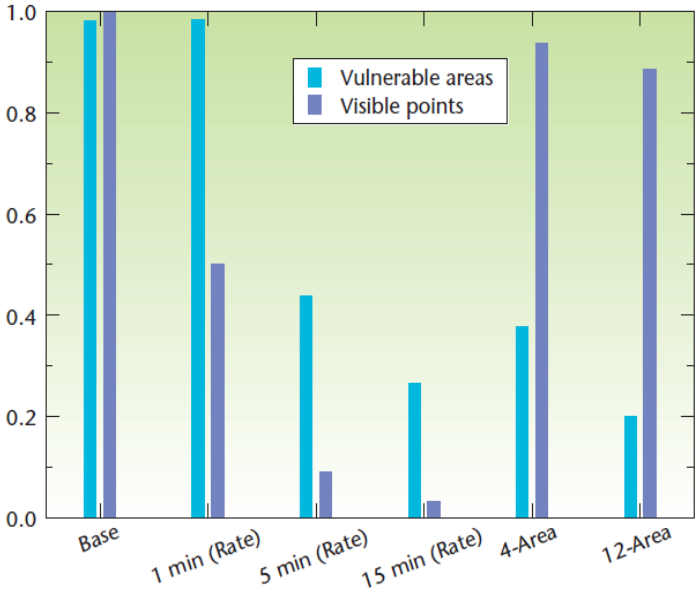


Abbildung 2.2: Ergebnisse aus [7]

Base-Algorithmus kann der Angreifer nahezu jedes sensitive Gebiet, das ein User betritt, erkennen. Bei der Bounded-Rate Version wird mit geringerer Updatefrequenz auch die Trefferrate geringer, allerdings auch die Genauigkeit in nichtsensitiven Gebieten. Das beste Ergebnis liefert die k-area Variante. Hier werden in nichtsensitiven über 90% aller Updates veröffentlicht, der Angreifer kann aber weniger als 40% der sensitiven Gebiete erkennen.
Dieser Ansatz versucht die Position eher zeitlich als räumlich zu verschleiern. Dies hat den Nachteil, dass ein Angreifer merkt, wenn sich der Benutzer gerade in einem sensitiven Gebiet befindet, da in diesem Fall die Positionsupdates ausbleiben bzw. weniger werden.

Protecting location privacy against spatial inferences: the PROBE approach

In [5] wird ein Ansatz vorgestellt, um eine Position auf Basis des geometrischen Kontexts und der Eigenschaften der Privatsphäre des Users zu verschleiern.
Der Benutzer definiert ein "privacy profile", das angibt wie sensitiv bestimmte Features(Kartenobjekte) für ihn sind. Dieses besteht aus einem Schwellenwert $v = T(ft)$ für jedes Feature ft.
Ein wichtiger Teil der Arbeit besteht daran eine Sensitivitätsmetrik zu definieren. Die Sensitivität eines Gebietes definiert, wie empfindlich dieses Gebiet für die Privatsphäre eines Benutzers ist. Die Sensitivität $P(r) = \int_r pdf$ sagt über die "probability density function" (pdf) (also eine Funktion die für einen Punkt die Aufenthaltswahrscheinlichkeit angibt) aus, wie wahrscheinlich der Benutzer sich im Gebiet r aufhält. Die "probability density function" ist für jeden Benutzer einzigartig und definiert sich über persönliche Merkmale wie Beruf oder Hobbies. So hat beispielsweise ein Arzt für ein Krankenhaus eine höhere Aufenthaltswahrscheinlichkeit als ein Büroangestellter. Formel 2.1 definiert die Sensitivität einer Region r bezüglich eines Featuretyps ft. $Cov(ft)$ ist die Fläche, die von Featuretyp ft abgedeckt wird. Die Sensitivität einer Region r bezüglich eines Featuretyps ft definiert sich durch die Aufenthaltswahrscheinlichkeit des Benutzers in der Region, die durch diesen Featuretyp innerhalb der Region r abgedeckt wird bezogen auf

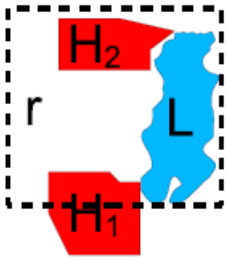


Abbildung 2.3: Beispiel einer sensitiven Region aus [5]. H1 und H2 stellen Krankenhäuser und L einen See dar.

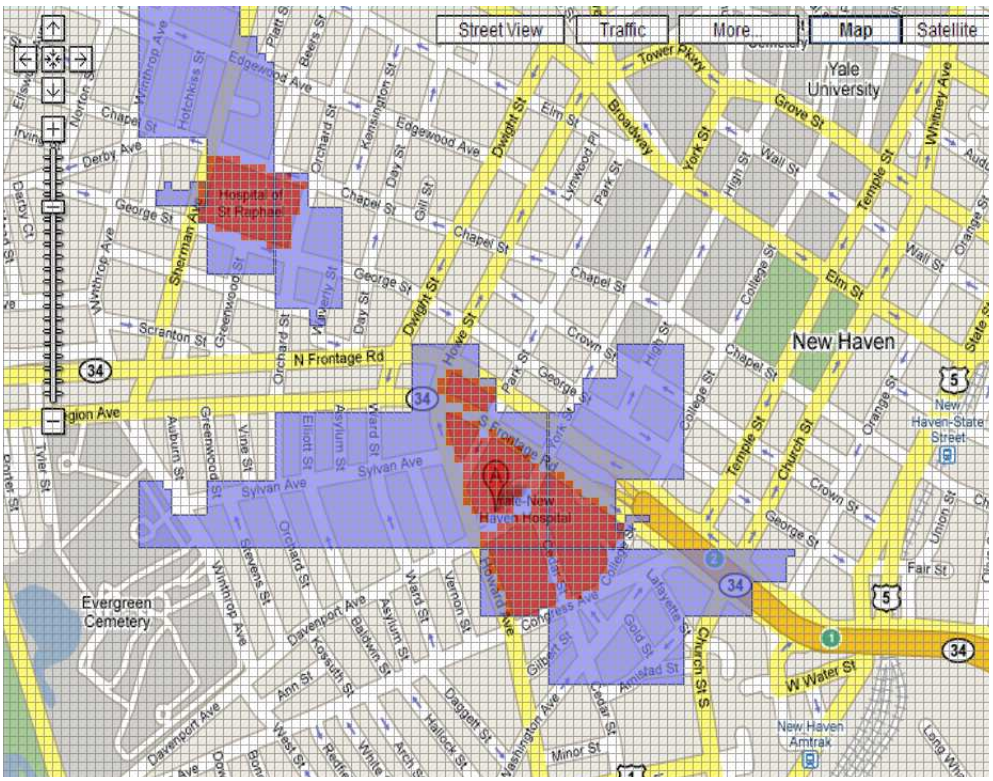


Abbildung 2.4: Beispiel für eine verschleierte Karte aus [5]

die allgemeine Aufenthaltswahrscheinlichkeit der Region.
Die Sensitivität einer Region r abhängig vom Featuretyp ft ist definiert durch:

$$P_{sens}(ft, r) = \begin{cases} \frac{\int_{Cov(ft) \cap r} pdf}{\int_r pdf} & wenn \int_r pdf \neq 0 \\ 0 & sonst \end{cases} \tag{2.1}$$

Das Verfahren ist ein zweistufiger Ansatz mit einer Offline- und einer Onlinephase. In der Offlinephase wird im Voraus eine verschleierte Karte anhand der Privatsphäre berechnet, in der sich überlappungsfreie, verschleierte Orte (obfuscated Locations) befinden. Diese werden in der Onlinephase nur noch aus der Karte abgefragt und an den Location Based Service (LBS) gesendet.
Die zu berechnende Karte wird mit einem Gitter aufgeteilt und die einzelnen Zellen werden nach dem Muster einer Hilbertkurve durchgegangen. Für jede Zelle (deckt Region r ab), deren Sensitivität $P(ft, r)$ für ein Feature ft den Schwellenwert $T(ft)$ überschreitet, wird die verschleierte Location dieser Zellen Schritt für Schritt auf die Nachbarzellen ausgeweitet (entlang der Hilbertkurve), bis der Schwellenwert eingehalten wird.

So entstehen größere Gebiete (obfuscated Locations), die durch Intervalle auf der Hilbertkurve definiert sind. Die Position eines Users kann einfach auf eine Zelle der Karte gemappt werden. Die so erstellten Gebiete (in Abbildung 2.4 oder 2.5 blau dargestellt) dienen als Lookup (effizient implementierbar als Quadtree) für die Zellen. Wird eine Zelle von einer verschleierten Position überdeckt, so wird als Position das Gebiet durch ein entsprechendes Intervall der Hilbertkurve preisgegeben. Ansonsten wird die einzelne Zelle (kleinstmögliches Intervall) veröffentlicht.

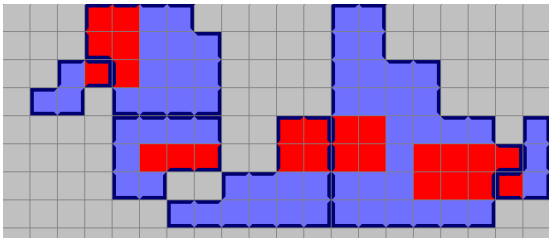


Abbildung 2.5: Verschleierte Regionen = lila, sensitive Features = rot [5]

In diesem Ansatz wird, im Gegensatz zum vorherigen, nicht die Anzahl der Positionsupdates über die Zeit beschränkt, sondern die Größe des möglichen Aufenthaltsgebietes in sensitiven Gebieten wird verändert. In diesem Fall hat das den Nachteil, dass komplexe Vorberechnungen nötig sind und aufgrund des Overlays mit einer Hilbertkurve ein begrenztes Gebiet erforderlich ist. Zudem muss der Nutzer einem Locationserver vollkommen vertrauen und kann nicht für verschiedene Dienste unterschiedliche Genauigkeiten festlegen.

Landscape-aware location-privacy protection in location-based services

Die Autoren von [4] schlagen einen sehr allgemeingültigen Ansatz vor und behandeln dabei sehr ausführlich und formal die Möglichkeiten, wie und unter welchen Bedingungen ein Angriff auf die verschleierte Position erfolgreich ist. Die Architektur, die hier zugrunde gelegt wird, sieht ähnlich aus wie beim in [7] beschriebenen Ansatz. Es gibt einen Benutzer A mit einem mobilen Gerät und einen Agenten B, der Positionsangaben des Benutzers verschleiert und an eine dritte Partei C wie z.B. einen LBS weitergibt. Dabei ist der Agent gegenüber dem LBS fair, d.h. er versorgt den LBS nicht mit falschen, sondern nur mit ungenauen Positionsinformationen. Diese sind entweder Kreise mit Radius R oder Quadrate mit Kantenlänge $2R$ je nach Koordinatensystem und Metrik. Für neutrale Landschaften (der Benutzer ist überall mit gleicher Wahrscheinlichkeit anzutreffen) reicht es die hinteren Stellen der echten Koordinaten abzuschneiden. Der Benutzter gibt gegenüber dem Agenten eine Wahrscheinlichkeit $p_{SLA/A}$an, mit welcher er durch einen zufälligen Angriff unentdeckt bleiben will. Davon hängt die zu wählende Größe R der

verschleierten Positionsinformation, die der Agent wählen muss, ab.
Es gilt folgende Gleichung zu erfüllen:

$$Pr(A\ and\ C\ within\ r) = \int_S dt\ Pr(A\ at\ t|B) \times \int_{\Omega_{(t,r)}} dz\ Pr(C\ at\ z|A\ atz) \leq p_{SLA/A} \tag{2.2}$$

Wobei $Pr(A\ and\ C\ within\ r)$ die Wahrscheinlichkeit ist, dass C die Position von A mit einem Abstand kleiner r trifft. S ist der gesamte Koordinatenraum und $\Omega_{(t,r)}$ ist eine Umgebung um Punkt t mir Radius r. t ist die Position von A und z die Postion an der C versucht A zu finden.
In dieser Arbeit präsentieren die Autoren viele Grundlagen für die Positionsverschleierung anhand probabilistischer Konzepte. Sie berücksichtigen unter anderem Messungenauigkeiten, verschiedene Koordinatensysteme sowie neutrale und nicht-neutrale Umgebungen.

2.2 Dateiformate

Im Bereich der Geoanwendungen haben sich verschiedene Datenformate etabliert. Hier sollen drei Formate näher betrachtet werden.

2.2.1 Geography Markup Language (GML)

GML ist ein auf XML aufbauendes Format, das vom Open Geospatial Consortium definiert wird, um Geodaten zu beschreiben [1]. Es soll ein offenes, herstellerunabhängiges Format zur Speicherung und Übertragung von Geodaten sein. Es unterstützt Profile mit eingeschränkter Funktionalität, damit Anwendungen nicht zwingend den gesamten Umfang unterstützen müssen. Es unterstützt verschiedene Primitive, wie z.B. Features, Geometrien, Topologien, Koordinaten-Referenzsystem oder Styling Rules. Es unterscheidet zwischen Features und geometrischen Objekten. Ein Feature ist ein physikalisches Objekt wie z.B. eine Straße, ein Gebäude oder auch ein Ort. Es kann, muss aber nicht, durch ein oder mehrere geometrische Objekte beschrieben werden. Die wichtigsten geometrischen Objekte sind Punkte, LineStrings und Polygone. Es gibt aber auch weitere Objekte wie z.B. Curves. Punkte sind durch ein Tupel definiert. Ein LineString ist eine Kurve, die durch mehrere Punkte definiert ist, zwischen denen die Kurve linear interpoliert wird. Polygone sind definiert durch äußere und innere Ringe. Ein Ring kann wie ein LineString definiert werden, dessen Anfang und Ende gleich sind und somit eine geschlossene Kette bilden. Äußere Ringe definieren die äußere Begrenzung eines Polygons, innere Ringe definieren "Löcher" in der Fläche eines Polygons.
GML bietet auch die Definition von Application Schemata. Schemata dienen zur Definition von Objekttypen. So können für verschiedene Anwendungen unterschiedliche Objekttypen definiert werden.

2.2.2 OpenStreetmap (OSM)

OpenStreetMap ist ein Projekt, welches, für jeden frei nutzbare, Geodaten sammelt und ähnlich wie Wikipedia, von der Community lebt. Für OpenStreetMap gibt es nicht nur ein Datenformat, sondern unterschiedliche Formate und Datenbanken für die Speicherung und Abfrage der Daten über eine API. Je nach Anwendungsgebiet gibt es verschiedene auf XML basierende Formate und einige Binärformate. Jedoch sind die geometrischen Objekte in allen Formaten die gleichen. Mögliche Objekte sind Punkte, Ways,

Closed Ways und Relationen. Ways entsprechen etwa den LineStrings bei GML und Closed Ways den Polygonen bei GML. Relationen sind Gruppierungen von andere Objekten. Dies kann eine Einfache Zusammenfassung von z.B. Polygonen sein um beispielsweise geographisch nicht zusammenhängende Flächen zu einer logischen Fläche zusammenzufügen. Jedes Objekt kann mit Tags versehen werden. Ein Tag ist ein key/value-Paar. Prinzipiell sind beliebige Schlüssel und Werte bei Tags möglich, jedoch sind von der Community für die meisten möglichen Kartenobjekte sinnvolle Schlüssel definiert.

2.2.3 Shapefile

Shapefiles [2] ist ein von ESRI Inc. (Environmental Systems Research Institute) entwickeltes Dateiformat für Geodaten. Es hat sich zu einem Quasi-Standard entwickelt und wird von einer Vielzahl an Programmen unterstützt. Entsprechend finden sich viele frei verfügbare Kartendaten im Shapefile Format. Gegenüber GML sind beim Shapefileformat die Daten über mehrere Dateien verteilt. Shapefiles ermöglichen es geometrische Objekte (Features) mit dazugehörigen Attributen in Form von Key/Value-Paaren zu speichern.
Die Dateien einer Shapefilesammlung sind:

- **.shp** - enthält die Geometriedaten
- **.dbf** - enthält Attribute zu den Geometriedaten in Form von key/value-Paaren
- **.shx** - verknüpft Attribute aus der .dbf-Datei mit den Geometriedaten aus der .shp-Datei.
- **.shp.xml** (optional) - Metadaten im XML-Format
- **.prj** (optional) - spezifiziert das Koordinatensystem

In Shapefiles sind 3 grundlegende Geometriearten möglich, die auf einer Ebene liegen. Durch weitere Eigenschaften lassen sich diese 3 Grundtypen um je 3 weitere Typen erweitern. Eine Erweiterung sind Multimengen. Z.B. kann ein Multipointobjekt mehrere Punkte enthalten, wird aber nur als ein Objekt behandelt. Eine weitere Möglichkeit sind Measured Shapetypes. Hier wird zusätzlich zu jedem Punkt zusätzlich zu den Koordinaten X und Y ein Wert M zugeordnet. Dieser kann für irgendwelche Werte wie beispielsweise eine Höhenangabe von Gebäuden genutzt werden. Eine zusätzliche Erweiterung ergibt sich durch eine zusätzliche Z-Koordinate, also Definition der Objekte

Geometrieart	Beschreibung
Point	definiert einen Punkt durch eine X- und eine Y-Koordinate
PolyLine	definiert eine Linie mit mehreren Abschnitten, die durch jeweils zwei Punkte gegeben sind. zusätzlich wird eine Bounding-Box (minX,minY, maxX, maxY) angegeben innerhalb welcher sich die Polyline befindet.
Polygon	definiert ein Polygon. Basiert auf den gleichen Angaben wie eine PolyLine, aber bildet mehrere geschlossene sich nicht überschneidende Ketten (Ringe). Die Reihenfolge der Punkte gegen den Uhrzeigersinn definiert hierbei Löcher (Inner Ring) innerhlab eines anderen Rings (Outer Ring).
Multi-...	definiert Multimengen einer der 3 Grundgeometrien. Es sind keine gemischten Multimengen möglich.
...Z	Wie vorhergehende Geometrien mit zusätzlicher Z-Koordinate
...M	Measured Shapetypes. Wie vorhergehende Typen mit einem zusätzlichen Wert.
Multipatch	Bildet eine Fläche aus mehreren Teilflächen. Teilflächen bestehen aus Triangle Strips, Triangle Fans oder Ringen

Tabelle 2.1: Geometriearten in Shapefiles

im Raum statt auf einer Ebene. Eine weitere Geometrieart ist der Multipatch, der eine Fläche aus mehreren Teilflächen bildet. Im Unterschied zu Polygonen können die Teilflächen hier auch durch “Triangle Strips” oder “Triangle Fans” beschrieben werden. Dies sind Flächen die durch nebeneinander oder im Kreis liegende Dreiecke gebildet werden.

Shapefiles haben in ihrer aktuellen Definition einen Nachteil, dass sie nur eine Art von Geometriedaten enthalten können. So besteht eine Karte in der mehrere verschiedene Geometriearten vorkommen aus mehreren Shapefiles. Zusätzlich werden die Daten gleicher Geometriearten nach semantischen Eigenschaften getrennt in verschiedene Shapefiles gespeichert.

2.2.4 Beispiele

Es gibt eine Vielzahl von Kartenmaterial in Form von Shapefiles. Hier sollen einige Beispiele gezeigt werden.

Datensatz	Inhalt	Shapetyp	Attribute	Attributwerte Beispiele
All Edges	Alle als Linien darstellbare Objekte	Polyline	FULLNAME ROADFLG RAILFLG HYDROFLG OLFLG	Green Lane 1/0 definiert jeweils ob eine Linie eine Straße, Bahnstrecke, Fluss/ Kanal oder Pipeline ist
Urbanized Areas	besiedelte Gebiete	Multipolygon	NAME00	Orlando,FL
Military Installations	Militärgelände	Multipolygon	FULLNAME	Naval Air Station Jacksonville
American Indian/ Alaska Native/ Native Hawaiian Area	Reservate für Ureinwohner	Multipolygon	NAME NAMELSAD	Miccosukee Miccosukee Reservation

Tabelle 2.2: Beispiele für Attribute in Kartendaten des U.S. Census Bureau

U.S. Census Bureau

Die U.S. Census Behörde bietet diverse Kartendaten der USA an. die mit statischtischen Daten verknüpft werden können. Die Daten sind gebündelt eines Staates oder Countys verfügbar.
Es sind unter anderem die in Tabelle 2.2 Datensätze verfügbar. Ein Datensatz wird von einer Shapefile repräsentiert. Die Datensätze auf statische Zwecke ausgerichtet. So sind Straßen als Linien modelliert und enthalten Attribute wie beispielsweise die Range der Hausnummern die sich Links und Rechts an einem Abschnitt befinden. An flächigen Modellierungen gibt es hauptsächlich Verwaltungsbezirke oder Stadtgebiete, aber keine genaueren Modellierungen von Häusern oder genauere Typisierung wie z.B. Krankenhäuser.

geofabrik.de

geofabrik.de bietet Daten des OSM-Projektes in Form von Shapefiles an. Es gibt Dateien von verschiedenen Regionen. Gesamte Kontinente, Länder oder nur einzelne Bundesländer in einer Dateisammlung. Die Qualität des Kartenmaterials ist stark abhängig von der Region, da sie von den freiwilligen Leistungen der Openstreemapsautoren abhängig sind. So sind Ballungszentren in Deutschland recht gut abgebildet, ländliche Gebiete und

andere Länder mit kleinerer Community sind nur sehr grob oder garnicht vorhanden. Die einzelnen Gebiete sind in verschiedene Pakete (.zip-Datei) aufgeteilt. Ein Paket enthält die in Tabelle 2.3 dargestellten Datensätze, die jeweils in einer Shapefile enthalten sind.

Datensatz	Inhalt	Shapetyp	Attribut	Attributwerte Beispiele
buildings	Gebäude	Multipolygon	name	Hauptbahnhof
			type	station, hospital
natural	natürliche Gebiete	Multipolygon	name	Schlosspark
			type	park, water
places	Orte	Point	name	Stuttgart
			type	suburb, village, ...
points	Interressante Orte	Point	name	Autobahnkreuz Ulm/ Elchingen
			type	station, motorway-junction, bus_stop, restaurant
Railways	Eisenbahnstrecken	Polyline	name	U-Bahnline U1
roads	Straßen	Polyline	name	Tübinger Straße
			type	secondary, ..
			oneway	1/0
			bridge	1/0
			maxspeed	50
waterways	Flüsse, Wasserstraßen	Polyline	name	Rhein, Panamakanal
			type	river, stream, canal

Tabelle 2.3: Beispiele für Attribute in Kartenmaterial von geofabrik.de

2.3 Analyse

Für die Speicherung von Geodaten bieten die XML-basierten Formate (OSM und GML) etwas mehr Möglichkeiten als Shapefiles. Da sie auf XML basieren, sind sie auch von Menschen lesbar, benötigen aber mehr Speicherplatz als das binäre Shapefileformat. Die grundlegenden Featuretypen wie Punkte, Linien und Polygone sowie die Gruppierung gleicher Basistypen beherrschen alle drei Formate. Das OSM-Format bietet zusätzlich

SHP	**OSM**	**GML**
Binärformat	XML-/Binärformat	XML-Format
Objekttypen		
Point	Node	Point
PolyLine	Way	LineString
Polygon	Closed Way	Polygon
Attribute	Tags	Properties

Tabelle 2.4: Vergleich der Datenformate

z.B. Relationen, mit denen Features unterschiedlicher Art zu einem neuen Feature gruppiert werden können. Das GML-Format kann zusätzlich mit komplexeren Typen wie Curves aufwarten, die komplexere Linientypen als die lineare Interpolation zwischen zwei Punkten bieten.
Die GeoTools Bibliothek bietet Funktionen zum Auslesen und Speichern von GML-Dateien und Shapefiles.
Bei den frei verfügbaren Datensätzen im Shapefileformat stechen die Daten des U.S. Census Bureau und von OpenStreetMap hervor, wobei hier die OSM-Daten eine größere Bandbreite an unterschiedlichen Featuretypen bieten. Während sich die Daten des U.S. Census Bureau hauptsächlich auf größere Flächen, wie Regierungsbezirke, sowie Straßen von in Form von Linien beschränkt, enthalten die OSM-Daten auch Flächen in kleinerem Maßstab, wie z.B. Gebäude oder Parks. Straßen sind hier jedoch auch hier nur als Linien modelliert. Für die Darstellung von featurebezogenen Aufenthaltswahrscheinlichkeiten bieten die OSM-Daten recht gute Möglichkeiten. Jedoch sind hier auf der Negativseite die nicht immer flächendeckend vorhandenen Daten aufzuführen.

3 Konzepte

3.1 Systemmodell

Zur Wiederholung wird nochmal auf den Begriff des Features eingegangen, wie er hier verwendet wird, um ggf. unterschiedliche Bedeutungen aus den vorgestellten Arbeiten und Dateiformaten auszuschließen. Ein Feature ist ein geometrisches Objekt einer Karte, das die Objekte, welche die Karte darstellt, geometrisch modelliert. Jedes Feature hat Attribute, um die Eigenschaften des Objektes zu beschreiben. Im Allgemeinen kann ein Attribut jedes beliebige Objekt sein. In Verbindung mit Shapefiles und der hier dargestellten Anwendung, reichen für Attribute auch nicht komplexe Attribute (Integer, Float und String).
Als Basis haben wir einen Benutzer eines Location-Based-Service, der sich im Bereich einer featurebasierten Karte bewegt. Zu dieser Karte gibt es verschiedene Informationen über den Benutzungsgrad. Dies können z.B. allgemeine statistische Informationen sein, wie z.B. Autos pro Stunde auf einer Straße. Weitere mögliche Informationen können auf einen Benutzer bezogene Eigenschaften sein, wie sein Bewegungsprofil oder Präferenzen für die Verschleierung seiner Position. Alle diese Eigenschaften werden pro Feature definiert. In einem ersten Schritt sollen diese Informationen für einen Benutzer in einer einfachen Art dargestellt werden können. Die Karte soll eine einfache "True/False"-Karte sein, die angibt, welche Gebiete für den Benutzer sensitiv (true) sind und welche nicht (false). Sensitive Gebiete können im einfachen Fall solche sein, in denen der Benutzer prinzipiell lokalisiert werden kann. Für einen Fußgänger wären dies z.B. Gebäude oder Gehwege. Beispiele für Gebiete, in denen er nicht lokalisiert werden kann, sind Autobahnen oder Seen.
Allgemein ist ein Gebiet für einen Benutzer sensitiv, wenn er nicht möchte, dass Dritte erfahren, wenn er sich in diesem Gebiet aufhält. Gebiete können auch auch unterschiedlich sensitiv sein, was meistens von der Aufenthaltswahrscheinlichkeit in verschiedenen Gebieten abhängt. Deshalb wird in einem zweiten Schritt die Darstellung dahingehend

erweitert, dass Features, entsprechend der Aufenthaltswahrscheinlichkeit des Benutzers, mit verschiedenen Farbabstufungen eingefärbt werden können.

Objekttypen

Die in 2.2 betrachteten Dateiformate können zwar mit einer Vielzahl von Objekttypen umgehen, für die Visualisierung und Auswertung von Aufenthaltswahrscheinlichkeiten von Kartenobjekten eignen sich jedoch nicht alle. Da Linien und Punkte im Gegensatz zu Polygonen keine Fläche haben, sind sie nicht geeignet um Aufenthaltswahrscheinlichkeiten darzustellen, da jedes Objekt, an dem sich jemand aufhalten kann, eine Fläche hat. Meistens werden Straßen als reine Linien modelliert, da sie oft nur zur Routenplanung verwendet werden. Wenn relativ kleine Gebiete, wie z.B. ein Stadtgebiet, betrachtet werden, sollten Straßen auch als Polygone modelliert sein, da hier die Breite auch ins Gewicht fallen kann. Mit Polygonen können alle Objekte, die aus der Vogelperspektive betrachtet eine Fläche bilden, modelliert werden

Objekttyp	Verwendung
Point	weniger geeignet zur Modellierung von Objekten.
Line	Straßen bei kleinen Maßstäben
Polygon	Gebäude, reale Flächen (Parks, Straßen, Plätze), logische Flächen (Regierungsbezirke, Naturschutzgebiete)

Tabelle 3.1: Mögliche Objekttypen und Anwendung

3.2 Anwendung für das Position Sharing Verfahren

Diese Anwendung ist in Verbindung mit dem Position Sharing Verfahren [6] entstanden. Dieses Verfahren erzeugt verschleierte Positionsinformationen mit unterschiedlicher Genauigkeit für verschiedene Location-Server. Die Positionsangabe (Share) besteht aus einem Kreis um einen Punkt, in dem sich der User befinden kann. Da unterschiedliche positionsbasierte Dienste unterschiedlich genaue Positionsangaben benötigen um ihre Dienstqualität zu erfüllen, aber der User nicht allen Diensten das gleiche Vertrauen entgegenbringt, müssen unterschiedliche Dienste mit unterschiedlich genauen Positionsinformationen versorgt werden. Dies erfolgt durch die Erzeugung von mehreren Shares,

die ineinander enthalten sind. Ein Share ist definiert durch einen Richtungsvektor der den Mittelpunkt relativ zum Mittelpunkt des nächstgrößeren Shares angibt. Der Mittelpunkt des ersten Shares ist absolut angegeben. Die Shares werden an unterschiedliche Locationserver geschickt, auf die dann die Dienste zugreifen können. Abhängig von ihrer Vertrauenswürdigkeit bekommen sie Zugriff auf unterschiedlich viele Shares aus denen sie eine genauere Position errechnen können. Der entwickelte Algorithmus läuft erst mal ohne Beachtung der Karteninformationen und damit der Aufenthaltswahrscheinlichkeiten des Benutzers ab. Für eine Erweiterung soll eine Visualisierung der verschleierten Regionen (Shares) auf einer featurebasierten Karte, eingefärbt in Abhängigkeit von Wahrscheinlichkeitswerten, ähnlich Abbildung 3.1 entstehen.

Abbildung 3.1: Featurebasierte Karte, eingefärbt anhand von Wahrscheinlichkeiten

Normalisierung

Es kann immer nur ein begrenztes Gebiet betrachtet werden bzw. es ist meistens nur ein bestimmtes Gebiet interessant. Typischerweise wird oft ein relativ abgeschlossenes Gebiet, wie z.B. eine Stadt, betrachtet. Dies ist beispielsweise der Fall, wenn ein Benutzer in einem Stadtteil wohnt und in einem anderen arbeitet. So bewegt er sich größtenteils innerhalb der Stadtgrenzen. Um sinnvolle Ergebnisse zu erhalten, sollten die Aufenthaltswahrscheinlichkeiten des Users auf das betrachtete Gebiet R normalisiert werden. Das heißt, die Aufenthaltswahrscheinlichkeiten müssen so angepasst werden, dass die

Wahrscheinlichkeit P für das gesamte betrachtete Gebiet $P(R) = 1$ ist. Dies kann erreicht werden indem Die Wahrscheinklichkeitswerte P aller Features x einer Region normalisiert werden:

$$\forall x \in R : P(x)_{norm} = \frac{P(x)}{\int_{y \in R} P(y)} \tag{3.1}$$

3.3 Überlappende Features

Je nach Modellierungsart kann es überlappende Features geben. Der Ansatz aus [5] geht beispielsweise von überschneidungsfreien Features aus. Dabei muss z.B. für ein Haus, das in einem Park steht, die Fläche des Parks ein Loch an der Stelle des Hauses haben. Das macht zwar die Modellierung einfacher, hat aber einige Einschränkungen. Es gibt zwei grundlegende Verwendungsmöglichkeiten für sich überlappende Features:
Zum einen die Einbeziehung der Höhe. So kann bei überschneidungsfreien Features bei Gebäuden nur eine Aussage über das gesamte Gebäude getroffen werden, aber nicht für einzelne Stockwerke.

Zum anderen konnten Features nicht nur real vorhandene Objekte abbilden, sondern auch organisatorische, Verwaltungs- oder logische Strukturen. So kann sich beispielsweise ein Haus gleichzeitig in einem Naturschutzgebiet und einem Landkreis befinden.
Je nach dargestellten Daten muss die Gesamtinformation von überlappenden Features unterschiedlich behandelt werden. Bei für die Features absoluten Werten, wie beispielsweise Autos pro Stunde auf einer Straße, lassen sich die Werte für eine Brücke, die über eine andere Straße führt, einfach addieren. Werden Aufenthaltswahrscheinlichkeiten dargestellt, dürfen die Wahrscheinlichkeiten zweier Features nicht einfach addiert werden, sondern müssen nach Formel 3.3

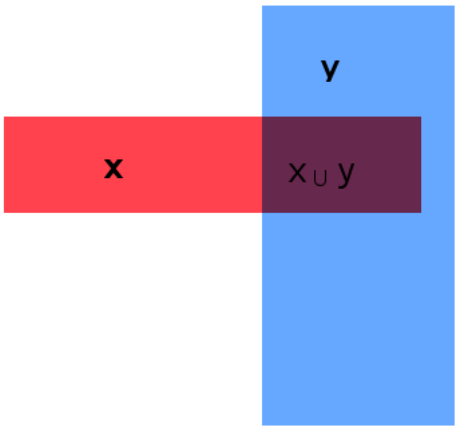


Abbildung 3.2: Zwei sich überlappende Features

berechnet werden. Hierbei sind $p(x) \cdot (1 - p(y))$ bzw. $p(y) \cdot (1 - p(x))$ die Wahrscheinlichkeiten, dass x ohne y bzw. y ohne x auftritt. $p(x) \cdot (y)$ ist die Wahrscheinlichkeit, dass beide auftreten. Addiert ergeben diese die Gesamtwahrscheinlichkeit. Wahrscheinlichkeit in einem Punkt für überlappende Features:

$$p(x \cup y) = p(x) \cdot (1 - p(y)) + p(y) \cdot (1 - p(x)) + p(x) \cdot p(y) \tag{3.2}$$

$$= p(x) + p(y) - 2 \cdot p(x)p(y) \tag{3.3}$$

3.4 Darstellung

Für die Erzeugung der Darstellung gibt es verschiedene Möglichkeiten: Zum einen ist es möglich einen einfachen "Pixel-Renderer" zu implementieren, der die Daten des anzuzeigenden Kartenausschnitts aus den vorgegebenen Datenstrukturen der GeoTools Bibliothek Pixel für Pixel ausliest. Dies hat den Vorteil, dass für jeden Punkt gleich alle Features an dieser Stelle direkt ausgewertet werden können und so z.B. die Überlappung von Features einfach berücksichtigt werden kann.
Der Renderer der GeoTools Bibliothek hat den Vorteil, dass er schon mit den Datenstrukturen der GeoTools Bibliothek zusammenarbeitet und ohne größere Arbeit schon viele Möglichkeiten unterstützt. So können Featurequellen zwischengespeichert und indiziert werden, um die Darstellung zu beschleunigen. Die Darstellung von sich überlappenden Features ist hier etwas komplizierter, da der Renderer jedes Featureobjekt für sich betrachtet. Der Renderer von GeoTools bietet ein fertiges Panel zum Einbinden in eine Java-Swing-Anwendung inklusive Steuerelemente wie z.B. für Zoom.

4 Implementierung

Dieses Kapitel beschreibt einige der verwendeten GeoTools Klassen und im Abschnitt 4.2 die Klassen und die wichtigsten Methoden der implementierten Anwendung.

4.1 Verwendete GeoTools Klassen

In diesem Abschnitt sollen die wichtigsten der verwendeten Klassen der GeoTools Biblothek beschrieben werden.

org.geotools.swing.JMapPane

Diese Klasse bietet ein Panel zum Anzeigen von Karten mit GeoTools innerhalb einer Java-Swing-Anwendung. Die Kartendaten erhält das Panel in Form eines MapContexts.

org.geotools.map.MapContext

Dies ist eine Klasse, welche den Inhalt einer Karte speichert. Dazu gehören zum einen mehrere Layer, die jeweils aus einer FeatureSource oder FeatureCollection ihre Daten beziehen. Zum anderen gehört zu einem MapContext auch die Angabe eines Koordinatenreferenzsytems (CoordinateReferenceSystem), das angibt wie die Koordinaten der Features definiert sind. Weiterhin enthält diese Klasse auch die Angabe der Area of Interest, also des Ausschnitts der Karte, der betrachtet wird.
Konkret wird die Subklasse org.geotools.map.DefaultMapContext verwendet, da sie nicht angegebene aber benötigte Werte selber errechnet. Dazu zählen z.B. die äußeren Abmessungen der Karte.

org.geotools.swing.action.PanAction/ZoomInAction/ZoomOutAction

Diese drei Klassen stellen Aktionen zur Verfügung, die in Verbindung mit einem JButton Zoom- und Verschiebefunktionen der Karte eines JMapPane bieten.

org.geotools.styling.Style

Eine Instanz der Style-Klasse kann zusammen mit den Kartendaten an einen MapContext übergeben werden und definiert, wie diese Daten dargestellt werden. Für jeden Layer eines MapContext ist ein eigener Style möglich.
Die Definition der Darstellung selber geschieht über eine Liste von Regeln (org.geotools.styling.Rule), welche es ermöglichen Features nach verschiedenen Kriterien auszuwählen und zu gestalten. Die Features, die von keiner Regel betroffen sind, werden durch eine Defaultregel dargestellt.

org.geotools.styling.Rule

Diese Klasse definiert eine Regel, wie eine Auswahl von Features dargestellt wird. Sie beinhaltet einen Filter, der definiert auf welche Features die Regel angewendet wird. Die Gestaltung der Features übernimmt ein Symbolizer, welcher es z.B. ermöglicht für Linien (sowohl Linien als Feature, als auch Linien als Umrandung von Polygonen) Stil, Dicke oder Farbe zu verändern. Genauso lassen sich auch bei Polygonen die Eigenschaften der Fläche in Farbe oder Transparenz verändern.

org.opengis.filter.Filter

Dies stellt ein Interface für Filterklassen dar. Die implementierenden Klassen von GeoTools bieten eine Vielzahl an Möglichkeiten, Features in verschiedenen Anwendungsfällen z.B. aus einer FeatureCollection zu selektieren. Es existieren verschiedene Gruppen von Filtern:

- **logische Filter**: Logische Filter sind die einzigen Filter, die andere Filter enthalten können. Mit ihnen können andere Filter logisch verknüpft werden.
- **geometrische Filter**: Diese Filtergruppe kann auf geometrischer Ebene Operationen wie Überlappung, Schnitt oder Beinhaltung geometrischer Objekte durchführen
- **Attribut Filter**: Filter dieser Gruppe können die Attribute von Features miteinander oder mit festen Werten vergleichen.

org.geotools.data.FeatureSource

Diese Klasse stellt eine einheitliche Schnittstelle zum Zugriff auf eine Featurequelle dar. Dies kann eine einfache Datei über FileDataStore oder auch eine Datenbank wie PostGIS. Diese Klasse bietet nur die Möglichkeit Daten zu lesen.

org.geotools.data.FileDataStore

Diese Klasse stellt ein Interface zu einer "physikalischen" Quelle von Features, wie z.B. eine Shapefile dar. Um auf die Features selber zuzugreifen bietet diese Klasse verschiedene Methoden zur Erzeugung eine FeatureSource-Instanz oder FeatureWriter-Instanz, die auch für Schreibzugriff genutzt werden kann.

org.opengis.feature.Feature

Dies stellt ein geographisches Objekt dar, das aus geographischen und nicht geographischen Eigenschaften besteht.

org.opengis.feature.simple.SimpleFeature

SimpleFeatures sind einfacher zu handhabende Features, allerdings mit Einschränkungen. Eine FeatureSource, die eine Schnittstelle zu einer Shapefile herstellt, liefert nur SimpleFeatures, da diese ausreichend sind um Features einer Shapefile zu beschreiben. Ein SimpleFeature ist in seinen Attributen eingeschränkt. Attribute können keine komplexen Datentypen sein. Attribute können nur einfach vorkommen und sind in ihrer Reihenfolge geordnet.

4.2 Klassendokumentation

Im Folgenden werden die erstellten Klassen und die wichtigsten ihrer Methoden dokumentiert.

MapviewerFrame

Diese Klasse stellt das Hauptfenster der Anwendung dar. Von hier aus werden alle Dialoge und Aktionen ausgelöst. Sie erstellt den Großteil aller grafischen Elemente der Anwendungsoberfläche und verknüpft diese mit den zugehörigen Funktionen und der

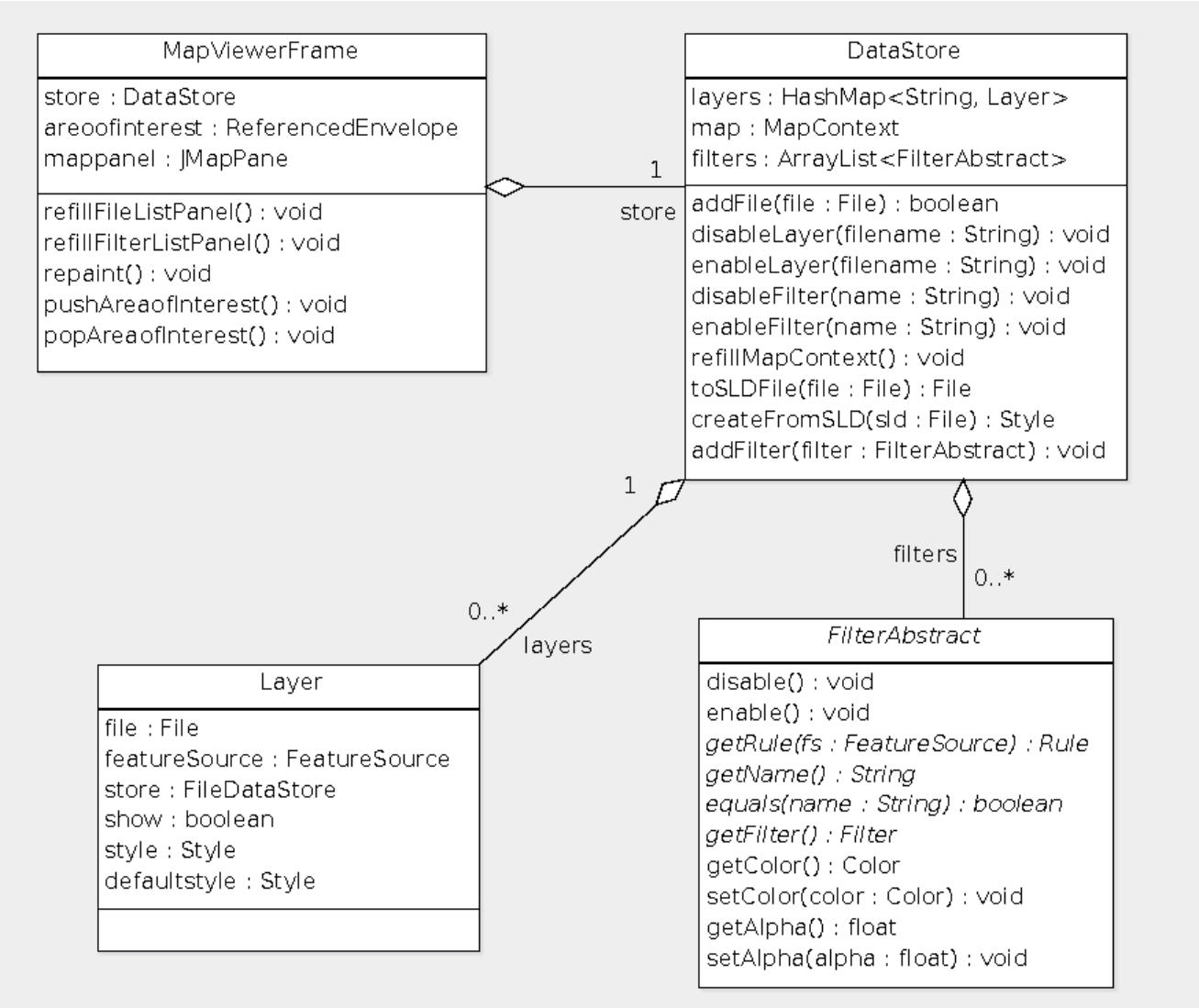


Abbildung 4.1: UML-Klassendiagramm der wichtigsten Klassen

DataStore-Klasse. Neben den Variablen für die Fenstergestaltung (GUI) hat diese Klasse nur zwei Variablen: Zum einen eine Instanz der DataStore-Klasse und zum anderen die AreaofInterest, ein Rechteck, das den gerade betrachteten Kartenausschnitt definiert. Unter den Variablen, die für die Fenstergestaltung zuständig sind ist noch eine Instanz von org.geotools.swing.JMapPane interessant. Sie stellt die eigentliche Karte dar.
Die wichtigsten Methoden dieser Klasse sind:

- void **pushAreaofInterest()** : Speichert den aktuellen angezeigten Kartenabschnitt in einer Variable ab. Wird beim Aktualisieren der MapContexts benötigt, da die Ansicht wieder auf die ursprüngliche zurückgesetzt wird.

- void **popAreaofInterest()** : Setzt den gespeicherten Kartenabschnitt auf den Mapcontext zurück.
- void **repaint()** : Zeichnet das Fenster und alle Kindelemente, insbesondere das Kartenpanel neu.
- void **refillFileListPanel()** : Aktualisiert die Liste der offenen Dateien mit den Daten aus dem DataStore.
- void **refillFilterListPanel()** : Aktualisiert die Liste der vorhandenen Filter mit den Daten aus dem DataStore.

Dazu kommen noch weitere Methoden, welche zum Auslösen der verschiedenen Funktionen und als Callback der einzelnen Dialoge dienen.

DataStore

Die DataStore-Klasse stellt einen Datenspeicher für die Kartendaten dar. Hauptbestandteile sind der MapContext, eine Liste der geöffneten Dateien und eine Liste mit Filtern. Der MapContext enthält die aktuell aktiven Layer und Filter und dient der Hauptanwendung um die Karte darzustellen.
Die Liste der geöffneten Dateien ist als Hashmap mit den Dateinamen als Indices implementiert. Die hier abgelegten Elemente sind DataStore.Layer-Objekte, welche alle zu einer Datei zugeordneten Daten enthalten.

- HashMap<String, Layer> **layers** : Enthält Objekte vom Typ DataStore.Layer die mit dem Dateinamen, der die Daten für den Layer liefert indiziert wird.
- MapContext **map** : Die MapContext-Instanz welche die aktuellen Daten, die angezeigt werden sollen, enthält.
- ArrayList<FilterAbstract> filters : Eine Liste von FilterAbstract-Objekten, die in der Anwendung aktiv sind.

Methoden:

- boolean **addFile(File file)** : Fügt eine weitere Datei als neuen Layer hinzu.
- void **disableLayer(String filename)** : Deaktiviert einen Layer/Datei.

- void **enableLayer(String filename)** : Aktiviert einen Layer/Datei.
- void **disableFilter(String name)** : Deaktiviert einen Filter
- void **enableFilter(String name)** : Aktiviert einen Filter.
- void **refillMapContext()** : Aktualisiert den MapContext mit den aktuell aktivierten Filtern und Layern. Zuerst werden alle Layer des MapContexts entfernt. Dann wird für jeden einzelen Layer, der in der HashMap "layers" gespeichert ist überprüft ob dieser aktiv (Layer.show ==true) ist. Wenn ja wird dieser zusammen mit dem Style, der von buildStyleFromFilters() erzeugt wird, dem MapContext hinzugefügt.
- File **toSLDFile(File file)** : Speichert einen Style in einer .sld Datei
- Style **createFromSLD(File sld)** : Lädt einen Style aus einer .sld Datei
- void **addFilterEquals(String attr, String value , Color color)** : Fügt einen neuen Vergleichsfilter hinzu.
- void **addFilter(FilterAbstract filter)** : Fügt einen neuen Filter hinzu
- Style **buildStyleFromFilters(DataStore.Layer layer)** : Erzeugt für einen Layer einen Style aus den aktivierten Filtern. Wird von refillMapContext() verwendet.
- Rule **createRule(Color colorfill, Color colorstroke, FeatureSource fs)** : Erzeugt eine Rule-Objekt mit den Farben "colorfill" als Flächenfüllfarbe und colorstroke als Linienfarbe für den Featuretyp, der in FeatureSource fs gegeben ist.
- void **createshadingsteps(String attr, float min, float max, int steps)** : Erzeugt mehrere Filter, die eine anhand eines Attributwertes eine Farbabstufung erzeugen und legt diese in der filters-Variable ab.

DataStore.Layer

Die Layer Klasse innerhalb des Datastores dient nur der Gruppierung aller Variablen/ Objekte die logisch zu einer Datei gehören. Deshalb sind alle Variablen mit public Zugriff und es gibt keine weiteren Getter- und Setter-Methoden.

- File **file** : Der Filehandle zur geöffneten Datei

- FeatureSource **featureSource** : Das FeatureSource-Objekt zur geöffneten Datei.
- FileDataStore **store** : Das FileDataStore-Objekt zur geöffneten Datei.
- boolean **show** : Definiert ob dieser Layer angezeigt wird oder deaktivert ist.
- Style **defaultstyle** : Enthält den Style aus der dem Layer zugeordneten Datei, der standardmäßig verwendet wird.
- Style **style** : Enthält den auf den Layer angepasst Style, der durch Filter definiert wird.

FilterAbstract

Die Idee dieser Klasse ist es, Filter der GeoTools Bibliothek mit einer Farbe zu verknüpfen, in welcher Features, auf die der entsprechende Filter zutrifft, eingefärbt werden. Für jeden Filtertyp wird von dieser Klasse abgeleitet und eine eigenständige Klasse erstellt. In der Basisimplementierung wird auf komplexere Gestaltungsmöglichkeiten verzichtet. Es ist lediglich möglich eine Farbe zu definieren, in der alle Aspekte der Featuretypen eingefärbt werden. Dies ist hier auch in den meisten Fällen ausreichend.

- void **disable()** : Aktiviert Filter
- void **enable()** : Deaktiviert Filter
- String **getName()** : Gibt eine Kurzbezeichnung des Filters zurück, welche auch die Filtereigenschaften enthalten sollte.
- boolean **equals(String name)** : Vergleicht ob der übergebene String mit getName() übereinstimmt.
- void **setColor(Color color)** : Setzt die Farbe des Filters. In dieser Farbe werden die für Features,die den Eigenschaften des Filters entsprechen, eingefärbt.
- Color **getColor()** : Gibt die Farbe des Filters zurück.
- Rule **getRule(FeatureSource fs)** : Erzeugt passend zum FeatureTyp der angegebenen Featuresource ein Rule-Objekt.

Es gibt drei Implementierungen dieser Klasse:

- **FilterEquals** : Dieser Filter definiert einen Attributnamen und einen dazugehörigen Wert. Er trifft auf alle Features zu, die dieses Attribut mit diesem Wert enthalten.
- **FilterBetween** : Dieser Filter definiert statt einem genauen Wert einen Wertebereich auf den ein Attribut eines Features überprüft wird.
- **FilterWrapper** : Diese Implementierung hat keine konkrete Filterfunktion, sondern dient nur als Wrapper für beliebige GeoTools-Filter um in der FilterImport-Klasse beliebige Filter angeben zu können.

FilterImport

Diese Klasse dient als Schnittstelle zu Dateien, die Filterobjekte welche die FilterAbstract-Klasse implementieren. Sie dient einerseits dazu die Filterliste, die sich in der Anwendung erstellen lässt, zu speichern und wieder aufzurufen. Zusätzlich dient sie als Importschnittstelle, um fremde Daten in die Anwendung zu importieren. In der Datei selber lassen sich alle von GeoTools verfügbaren Filter erstellen, auch wenn es dafür keine konkrete Implementierung der FilterAbstract-Klasse gibt. In der Datei werden die Filter im XML-Format gespeichert. Eine genauere Beschreibung des Dateiformats findet sich in Kapitel 5.1.3.

- boolean **open(File file)** : Öffnet eine Datei.
- ArrayList<FilterAbstract> **importFilters()** : Gibt eine Liste von Filterobjekten, die in der geöffneten Datei enthalten sind zurück
- void **exportFilters(ArrayList<FilterAbstract> filterList)** : Speichert eine Liste von Filterobjekten

AttributeImport

Diese Klasse dient dem Import von zusätzlichen Attributen zu vorhandenen Features. Die übergebene Datei im XML-Format ist in zwei Abschnitte gegliedert: Der erste Abschnitt enthält die die Attributnamen und die Datentypen der Attribute. Der zweite Abschnitt besteht aus einer Liste von Datensätzen, die jeweils aus einem Filter und Attributen mit entsprechenden Werten bestehen. Der Filter bestimmt Bedingungen, unter denen einem

Feature die dazu angegebenen Attribute und Werte zugewiesen werden. Eine genauere Beschreibung des Dateiformats findet sich in Kapitel 5.1.4.

- boolean **importFile()** : Öffnet eine Datei mit Attributen.
- void **setToFeatureSource(FeatureSource fs)** : Fügt die Attribute der angegeben FeatureSource hinzu.

5 Evaluation

5.1 Anwendungsbeschreibung

5.1.1 Shapefile/Layer öffnen

Die Anwendung kann gleichzeitig mehrere Layer, die jeweils einer Shapefile entsprechen anzeigen.

Wird eine Datei geöffnet, so wird überprüft ob im gleichen Verzeichnis eine Datei mit dem gleichen Namen und der Erweiterung .sld vorhanden ist. Ist diese Datei vorhanden und enthält eine gültige Stylebeschreibung, so wird dieser Style verwendet um den Layer darzustellen. Ist keine .sld-Datei vorhanden, wird ein Dialogfenster geöffnet in dem die Anzeigeeinstellungen für den Layer definiert werden können. Hier lassen sich, je nachdem welcher Featuretyp in der Datei gespeichert ist, verschiedene Eigenschaften ändern. Abbildung 5.1 zeigt die Einstellmöglichkeiten für einen Polygonlayer. Es lassen sich Linien für die Umrandung sowie die Füllfarbe der Fläche verändern. Optional kann ein Attributsfeld als Text mit eingeblendet werden. Beim Featuretyp "Linie" fehlen die Optionen für die Füllfläche.

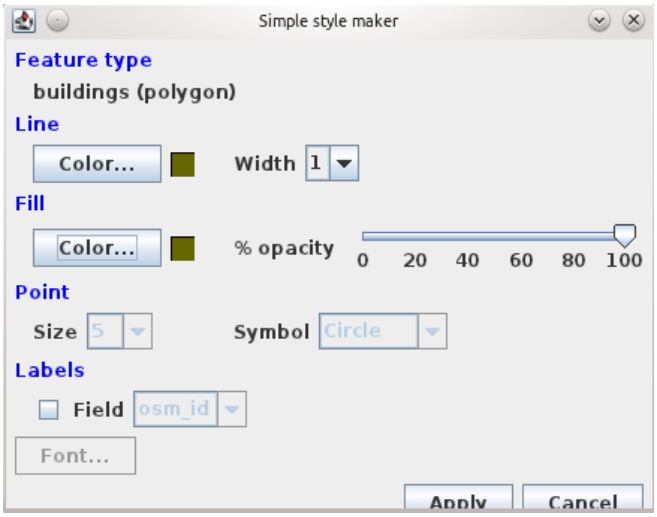


Abbildung 5.1: Styledialog

Filter	Funktion	Beispiel
Equals	Überprüft ein Attribut auf einen genauen Wert	type = station
Between	Feature liegt zwischen zwei Werten	0.1 < `probability` < 0.2

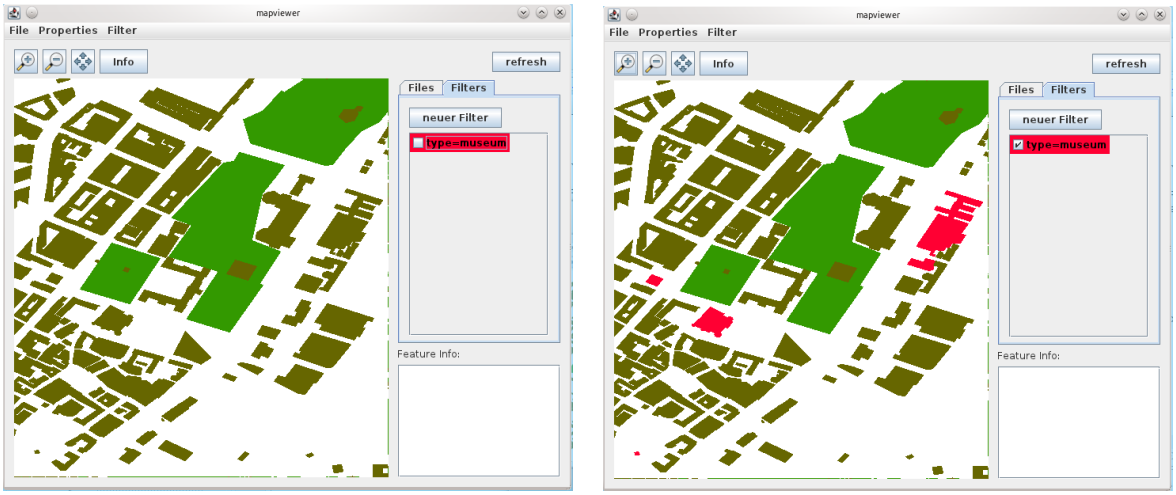
Tabelle 5.1: Mögliche Filter

5.1.2 Layer aktivieren/deaktivieren

Geöffnete Layer können, nachdem sie einmal geöffnet wurden, auch einzeln für die Anzeige deaktiviert werden, bleiben aber in ihren Einstellungen erhalten.

5.1.3 Filter

In der Anwendung lassen sich Kombinationen aus Filtern und Farbzuweisungen (hier kurz Filter genannt) erstellen, mit denen Features anhand ihrer Eigenschaften automatisch eingefärbt werden können. Trifft ein Filter auf ein Feature zu, so wird dieses in der dem Filter zugeordneten Farbe gefärbt. Die Features können anhand ihrer Attribute durch Anwendung der Filter in Tabelle 5.1 gefärbt werden.



(a) inaktiv (b) aktiv

Abbildung 5.2: Beispiel für die Darstellung von Features mit Filter

Filter Import/Export

Unter dem Menüpunkt “Filter” gibt es die Möglichkeit die erstellten Filter in eine Datei zu exportieren oder aus einer Datei zu importieren. So können erstellte Filterkonfigurationen gespeichert und wieder geladen werden. Es können auch Daten aus anderen Quellen importiert werden, indem man sie in das Datenformat überführt.
Das Dateiformat besteht aus einfachen XML. Eine Übersicht über alle möglichen Filter, die GeoTools zur Verfügung stellt findet sich unter http://docs.geotools.org/latest/javadocs/org/opengis/filter/Filter.html und unter http://schemas.opengis.net/filter/1.0.0/filter.xsd findet sich das entsprechende XML-Schema für die XML-Repräsentation der Filter. Exemplarisch sind im Beispielcode 5.1 zwei Filter dargestellt. Das Wurzelelement besteht aus einem Filter-Tag. Darin können beliebig viele Style-Blöcke enthalten sein, die alle jeweils einen Filter sowie ein Color-Tag und ein optionales Alpha-Tag beinhalten. Das Color-Tag definiert über seine Attribute “r”,“g”,“b” (rot-, grün-, blau-Anteil) in welcher Farbe ein Feature, auf das der zugeordnete Filter zutrifft, eingefärbt wird. Gültige Werte für die Attribute sind ganze Zahlen von 0 bis 255 und definieren zu welchem Anteil die jeweilige Grundfarbe einfließt. Das Alpha-Tag definiert eine Alphatransparenz. Ein Wert von 1 ist untransparent und ein Wert von 0 stellt volle Transparenz dar. Gültig sind hier Float-Werte von 0.0 bis 1.0 . Ist kein Alpha-Tag angegeben, so wird der Defaultwert 1.0 (keine Transparenz) genommen.

Listing 5.1: Beispiel für eine XML-Datei mit Filtern

```
<Filter>
        <Style>
                <PropertyIsEqualTo>
                        <PropertyName>type</PropertyName>
                        <Literal>water</Literal>
                </PropertyIsEqualTo>
                <Color r="204" g="51" b="0" />
                <Alpha value="0.5">
        </Style>
        <Style>
                <PropertyIsBetween>
                        <LowerBoundary>
                                <Literal>0.0</Literal>
                        </LowerBoundary>
                        <UpperBoundary>
```

```
                        <Literal>1.0</Literal>
                    </UpperBoundary>
                </PropertyIsBetween>
                <Color r="204" g="51" b="0" />
                <Alpha value="0.5">
            </Style>
            .....
</Filter>
```

Wahrscheinlichkeiten

Diese Filter dienen auch als Grundlage zur unterschiedlichen Einfärbung von Features mit unterschiedlichen Wahrscheinlichkeiten. Dafür gibt es zwei verschiedene Möglichkeiten: Über den Menüpunkt "Filter -> Shading" lassen sich automatisiert mehrere Filter erstellen. Unter Angabe eines Attributnamens, einem minimalen und einem maximalen Wert, einer Schrittweite sowie zwei Farbwerten erstellt diese Funktion eine Bandbreite von Filtern, die jeweils ein Intervall, entsprechend der Schrittweite, zwischen den beiden Werten abdecken und Features auf die dieser Filter zutrifft ein entsprechendes Farbintervall zuordnen. Hierfür müssen die entsprechenden Werte als Attribute in der darzustellenden Shapefile enthalten sein. Alternativ können die Werte schon vorher auf entsprechende Farben abgebildet werden und mit einem Filter den entsprechenden Features über eine eindeutige Id eines Features zugeordnet werden.

5.1.4 Import von Attributen

Eine wichtige Eigenschaft der Anwendung ist die Zuweisung von Aufenthaltswahrscheinlichkeiten für Features. Diese können entweder direkt in der Shapefile gegeben sein oder über eine XML-Datei importiert werden. Eine XML-Datei für den Import besteht aus zwei Sektionen: Aus der Definition der zu importierenden Datentypen und den Daten selber. Die oberste Ebene bildet das Wurzelelement "<Properties>...</Properties>". Dieses enthält den Definitionsabschnitt für die Variablentypen und die Werte der Variablen für bestimmte Features. Der Definitionsbereich enthält für alle zu importierenden Attribute ein <Attribute> Tag, welches mit dem Attribut "key" den Namen des Attributs definiert und dem Attribut "'type" den Typ. Mögliche Typen sind "Double", "Integer" oder "String". Die Werte selber sind innerhalb der "<set>...</set>"-Umgebung definiert.

Sie enthält zwei Arten von Tags: Innerhalb des <Filter> Tags wird definiert, welchen Features die Werte zugewiesen werden. Das <Attrib> Tag enthält die Werte. Wie beim Import der Filtereinstellungen kann auf alle Filter von GeoTools zurückgegriffen werden.

Listing 5.2: Beispiel einer XML-Datei für den Import von Attributen

```
<Properties>
        <Definition>
                <Attribute key="probability" type="Double" />
                <Attribute key="name" type="String" />
                ...
        </Definition>
        <set>
                <Filter>
                        <PropertyIsEqualTo>
                                <property>type</property>
                                <literal>train_station</literal>
                        </PropertyIsEqualTo>
                </Filter>
                <Attrib key="probability" value="0.33" />
                <Attrib key="name" value="Bahnhof" />
                ...
        </set>
        <set >
                ...
        </set>
        ...
</Properties>
```

5.2 Darstellungsergebnisse

Überlappende Features

Die Darstellung überlappender Features gliedert sich in zwei Fälle. Die Darstellung von absoluten Werten und die Darstellung von Wahrscheinlichkeiten. Für den ersten Fall können Features statt einer Farbabstufung der volle Farbwert zugewiesen und die Varianz der Farbe, abhängig vom Wert, durch Alphatransparenz realisiert werden. Dabei

muss darauf geachtet werden, dass die Transparenz entsprechend der überlappenden Features, die zusammen den größten Wert ergeben, skaliert wird. In Abbildung 5.3 wird ein Gebäude dargestellt, bei dem die zwei senkrechten Gebäudeteile über einem Keller gebaut sind. In dieser Karte könnte z.B. die Personendichte (Personen pro m^2) dargestellt werden. In den sich überlappenden Teilen herrscht somit eine höhere Personendichte, da dort auch mehrere Stockwerke existieren. Da hier die Transparenzwerte der einzelnen Features addiert werden, ist dies kein Problem.

Werden jedoch Wahrscheinlichkeiten dargestellt funktioniert nach Formel 3.3 dieser Ansatz nicht mehr. Da der Renderer der GeoTools-Bibliothek jedes Feature für sich und ohne Bezug zu seiner Umgebung darstellt, müssen die Daten anderweitig aufbereitet werden.

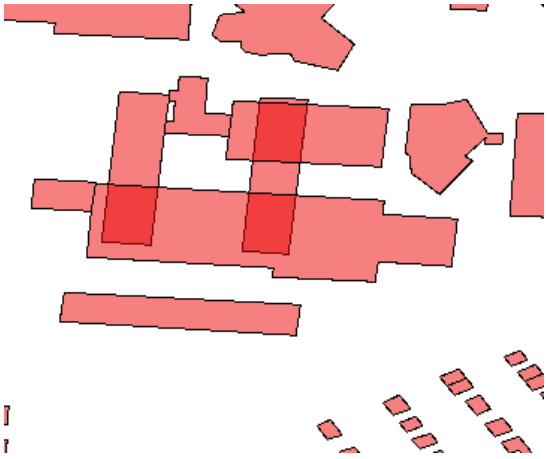


Abbildung 5.3: Darstellung überlappender Features

Eine mögliche Lösung wäre die Schnittfläche der sich überlappenden Features zu ermitteln, daraus entsprechende Features zu erstellen und die entsprechenden Werte nach Formel 3.3 zuzuweisen und diese in einem zusätzlichen Layer darzustellen. Dies muss entweder in einem Vorverarbeitungsschritt oder nur bei relativ kleinen Kartenausschnitten mit wenigen Features erfolgen, da im schlechtesten Fall jedes Feature mit jedem anderen verglichen werden muss, was einen erheblichen Rechenaufwand darstellt.

Klassische Darstellung

Abbildung 5.4 zeigt einen mit der Anwendung, auf Daten von OpenStreetMap basierend, erstellten Kartenausschnitt der Stadt Heilbronn. Dieser entspricht, stark vereinfacht, der üblichen Darstellung von Kartenmaterial wie beispielsweise in Google Maps. Die unterschiedlichen Einfärbungen sind durch Anwendung von Filtern entstanden. Zum Beispiel überprüft ein Filter das Attribut “type” auf den Wert “riverbank”, also einen Fluss und färbt Features, welche dem Filter entsprechen, blau ein. Entsprechend werden

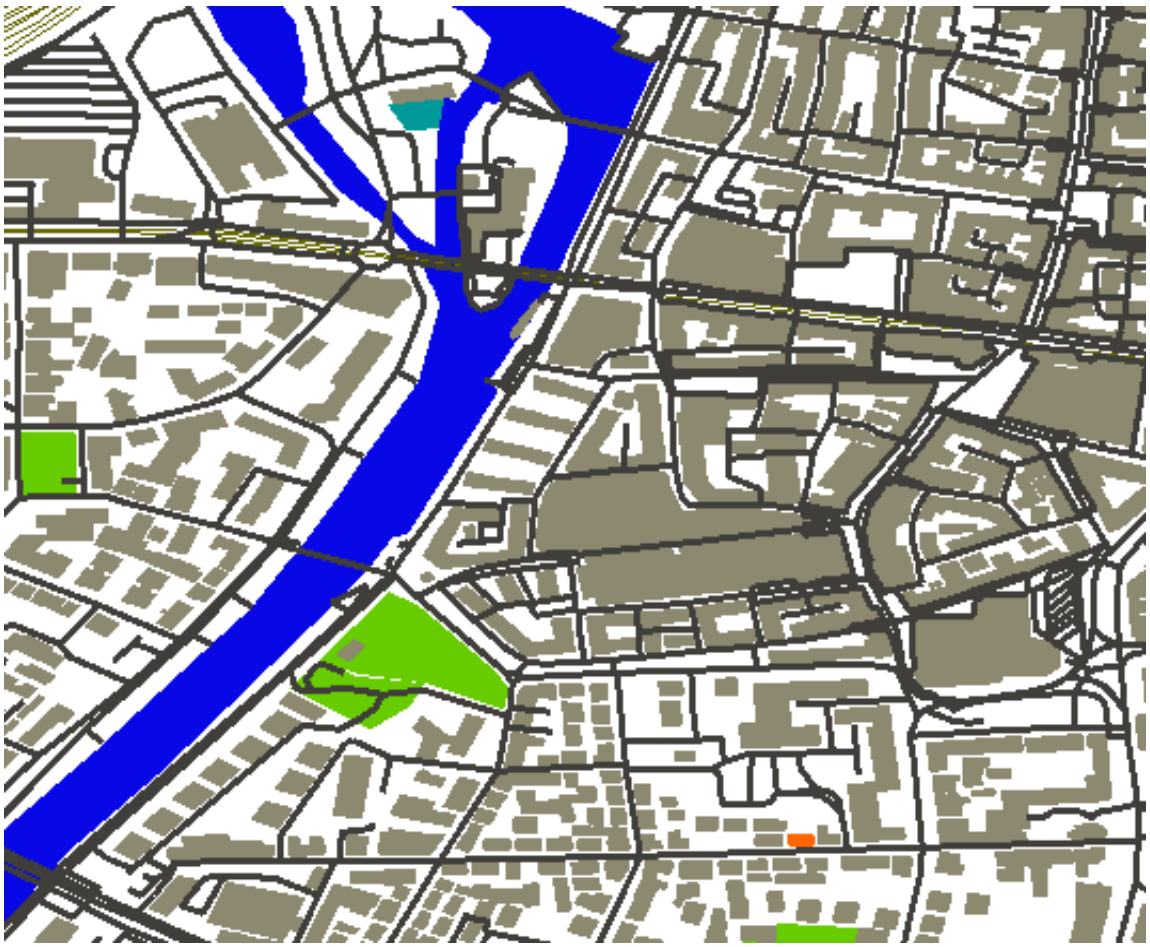


Abbildung 5.4: normale Kartendarstellung

Parks grün eingefärbt. In diesem Beispiel existiert noch ein Museum in türkis sowie eine Polizeistation in orange.

True/False Darstellung

Die True/False Darstellung in den Abbildungen 5.5 und 5.6 stellen dar, an welchen Orten Benutzer prinzipiell anzutreffen sind. Es wird in beiden Abbildungen der gleiche Ausschnitt wie in Abbildung 5.4 gezeigt. In beiden Fällen steht eine rote Färbung für "false", also Orte an denen der entsprechende Benutzer nicht anzutreffen ist und grün für "true" und damit Orte die ein Benutzer erreichen kann. Sowohl das Benutzerprofil für einen Fußgänger als auch einen Autofahrer erwartet den Benutzer logischerweise nicht in oder auf dem Fluss. Ein Fußgänger kann prinzipiell auf der Straße und in Gebäuden aufgefunden werden. Autofahrer hingegen sind in Gebäuden nicht zu erwarten.

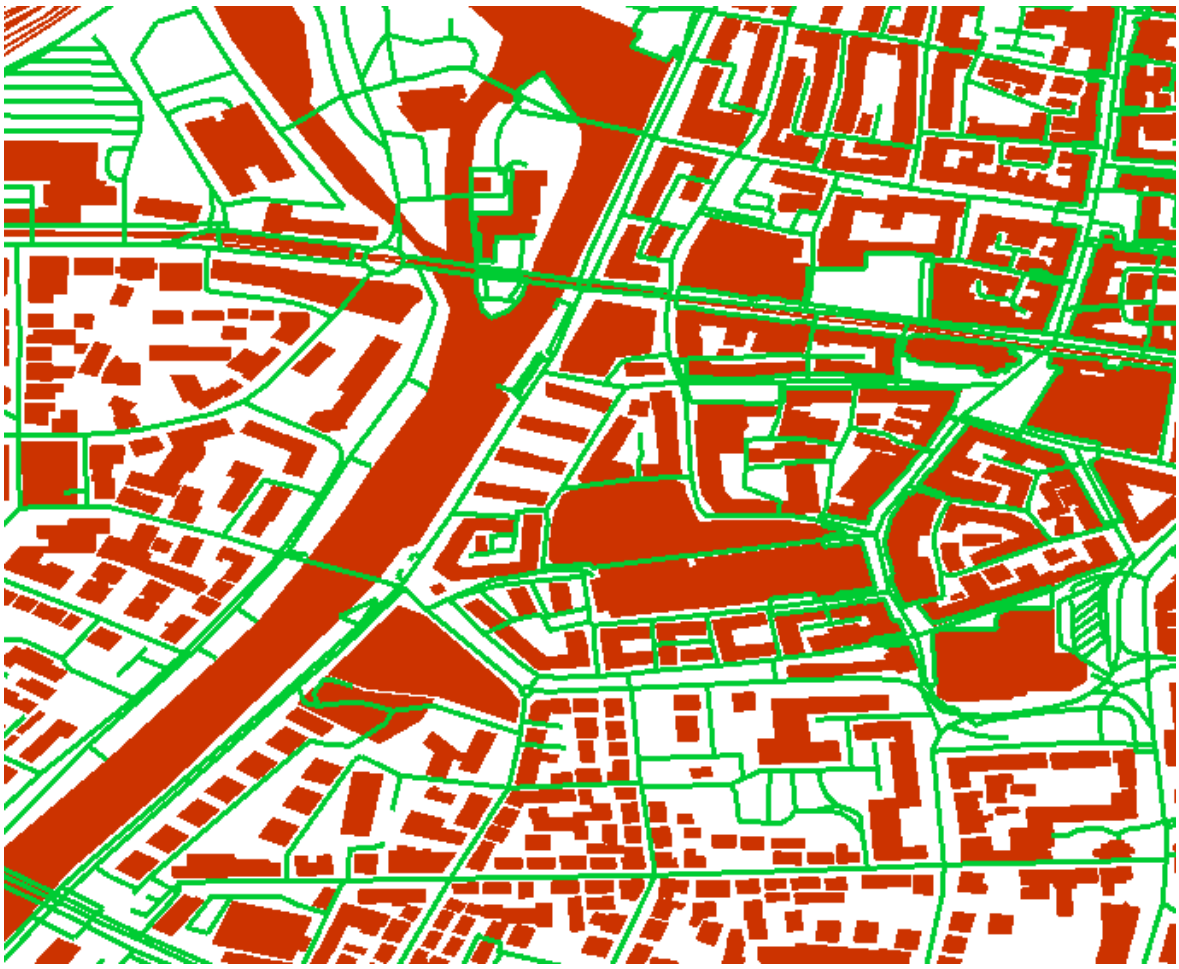


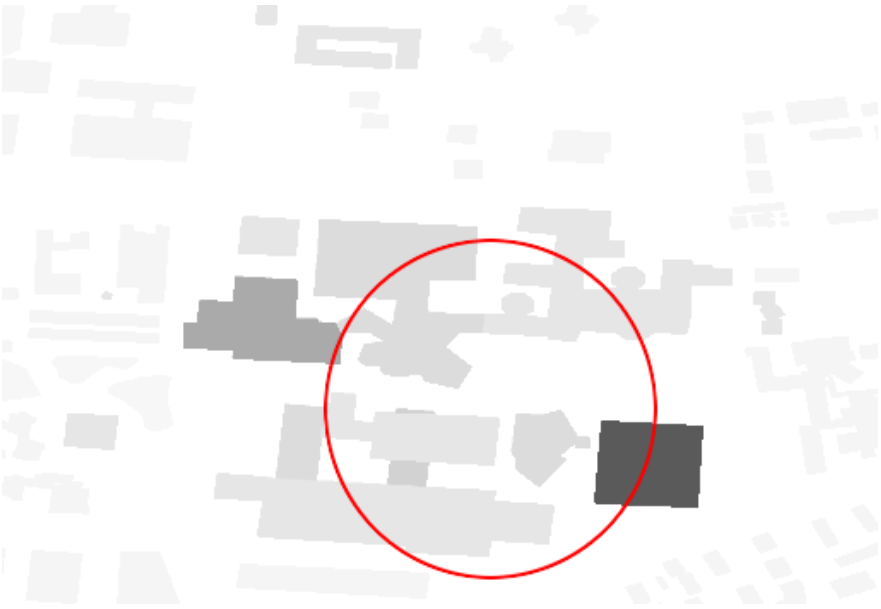
Abbildung 5.5: True/False-Darstellung für Autos

Darstellungen von Wahrscheinlichkeitsverteilungen

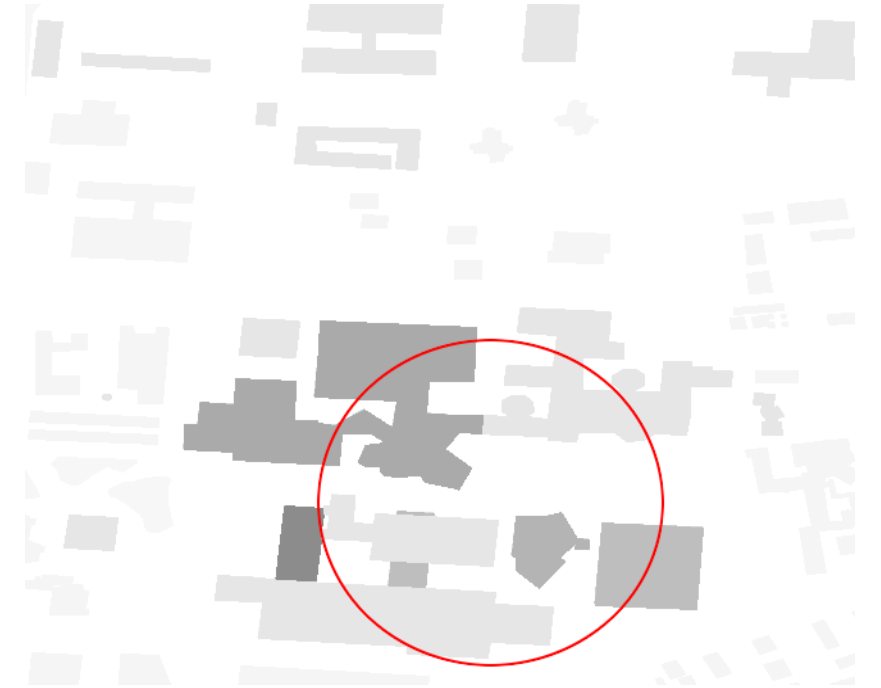
Abbildung 5.7 zeigt die Wahrscheinlichkeitsverteilung verschiedener fiktiver Personen für das gleiche Gebiet. Es handelt sich hier um eine Karte der Gebäude des Campus der Universität Stuttgart. Je dunkler die Einfärbung ist, desto höher ist für dieses Gebäude die Aufenthaltswahrscheinlichkeit des Benutzers. Man kann gut die Unterschiede im Verhalten erkennen. Der Mitarbeiter (Abb. 5.7(a)) der Universität hält sich die meiste Zeit in seinem Büro in dem Gebäude unten rechts auf. Einige weitere Gebäude auf dem Campus sind auch etwas dunkler gefärbt, da der Mitarbeiter dort auch Vorlesungen hält, oder in die Mensa geht. Der Student hingegen (Abb. 5.7(b)) verbringt die meiste Zeit auf dem Campus in unterschiedlichen Hörsälen. Bei ihm ist deshalb kein Gebäude erkennbar, in dem er sich größtenteils aufhält. Zur Verdeutlichung ist jeweils ein Kreis

Abbildung 5.6: True/False-Darstellung für Fußgänger

eingezeichnet, der eine verschleierte Position darstellen soll. Im Fall des Mitarbeiters ist es doch sehr wahrscheinlich, dass er sich innerhalb des Kreises ganz rechts aufhält. Im Fall des Studenten ist es nicht so klar, wo er sich aufhält.



(a) Mitarbeiter



(b) Student

Abbildung 5.7: Wahrscheinlichkeitsverteilung unterschiedlicher Personen

6 Zusammenfassung

Durch die Verbreitung von GPS-Empfängern und günstigen Datentarifen für mobile Geräte, drängen immer mehr Location Based Services auf den Markt. Da nicht jeder seine genauen Positionsinformationen preisgeben, aber trotzdem solche Dienste nutzen möchte, werden Möglichkeiten entwickelt, um Positionen zu verschleiern. Dabei spielen die Eigenschaften eines Benutzers eine wichtige Rolle. Dazu zählen beispielsweise die Bewegungsprofile und damit die Aufenthaltswahrscheinlichkeiten an verschiedenen Orten sowie die Präferenzen der Verschleierung.
Die hier entwickelte Anwendung bietet eine Möglichkeit diese Eigenschaften in Karten zu visualisieren. Mit Hilfe von Filtern bieten sich viele Möglichkeiten Kartenobjekte einzufärben und verschiedene Daten grafisch darzustellen.

Literaturverzeichnis

[1] Geography markup language. http://www.opengeospatial.org/standards/gml.

[2] Esri shapefile technical description. http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf, 1998.

[3] Abbildungen 1.1, 3.1, 5.2, 5.3, 5.4, 5.5, 5.6 und 5.7 sind mit Kartendaten von Openstreetmap entstanden. (c) 'OpenStreetMap' und Mitwirkende, CC-BY-SA. http://www.openstreetmap.org/, 2011.

[4] Claudio Agostino Ardagna, Marco Cremonini, and Gabriele Gianini. Landscape-aware location-privacy protection in location-based services. Journal of Systems Architecture, 55(4):243 – 254, 2009. Secure Service-Oriented Architectures (Special Issue on Secure SOA).

[5] Maria Luisa Damiani, Elisa Bertino, and Claudio Silvestri. Protecting location privacy against spatial inferences:the probe approach, 2009.

[6] Frank Dürr, Pavel Skvortsov, and Kurt Rothermel. Position Sharing for Location Privacy in Non-trusted Systems. In Proceedings of the 9th IEEE International Conference on Pervasive Computing and Communications (PerCom 2011), pages 189–196, Seattle, USA, März 2011. IEEE Computer Society.

[7] Marco Gruteser and Xuan Liu. Protecting privacy in continuous location-tracking applications. IEEE Security and Privacy, 2:28–34, 2004.

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

Andreas Paul:

Stuttgart, 31.05.2011