
Bachelorarbeit am Institut für Informatik der Freien Universität Berlin,

Arbeitsgruppe Netzbasierte Informationssysteme

# Ein Dienst zur automatischen Umwandlung von WMS-Kartenbildern zu textuellen Geodaten

Ahmet-Serdar Karakaya
Matrikelnummer: 4552318
ahmetserdar@hotmail.de

Erstgutachter: Prof. Dr. Adrian Pasche
Zweitgutachter: Dipl.-Inf. Marko Harasic

Berlin, 02.04.2015

*Ahmet-Serdar Karakaya*

**Zusammenfassung**

International akzeptierte Standards für Geoinformationssysteme ermöglichen es unter anderem, erwünschte Geodaten zu einem Thema zu finden. Für verschiedene Formate und Modelle von Geodaten existieren verschiedenen Standards, die das Finden und Benutzen der Geodaten ermöglichen. Eines dieser Standards ist das *Web Map Service*, der die Geodaten in Form von *Rasterdaten* anbietet. Rasterdaten sind aber nicht immer die bessere Wahl für Geodaten und so ist eine Methode nötig diese in Vektordaten umzuwandeln. Dieser Prozess kann manuell vollzogen werden, was aber zeitaufwändig und mühsam ist. Deshalb wird im Rahmen dieser Bachelorarbeit ein *Umwandlungsdienst* erstellt und vorgestellt, der diesen Prozess automatisieren soll. Anschließend wird erörtert, inwiefern der Umwandlungsdienst diesen Prozess übernehmen kann.

## Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, den 02.04.2015

Ahmet-Serdar Karakaya

# Inhaltsverzeichnis

# 1 Einleitung

Durch die steigende Beliebtheit von mobilen Geräten wie Smartphones und Tablet-PCs, steigt die Nachfrage an Applikationen für mobile Geräte. Somit wird die Frage nach dem Datenfluss zwischen den Applikationen und dem Internet immer wichtiger. Je größer die zu übermittelnden Dateien sind, desto länger dauert es die benötigten Dateien über das Internet zu senden bzw. zu empfangen. Ein weiterer Aspekt ist das Datenvolumen des Nutzers. Hat der Nutzer nämlich ein begrenztes Datenvolumen für die Kommunikation mit dem Internet, könnte er davon abgeschreckt werden Applikationen zu benutzen, die das Datenvolumen stark belasten. Aus diesem Grund sind Entwickler von Applikationen bemüht, den Datenvolumenverbrauch ihrer Applikation zu reduzieren. Eine Möglichkeit das zu erreichen ist die Applikation möglichst selten mit dem Internet verbinden zu lassen und die empfangenen Dateien auf dem Gerät zu speichern, sodass diese Daten nicht erneut heruntergeladen werden müssen, falls sie wieder gebraucht werden. Eine andere Möglichkeit den Datenvolumenverbrauch einer Applikation zu verringern ist es, die zu übertragenden Dateien möglichst klein zu halten. Dazu können die zu übermittelnden Dateien vor dem Abschicken komprimiert werden, was den Nachteil hätte, diese dann nach dem Empfangen dekomprimieren zu müssen, wodurch die Applikation längere Wartezeiten hätte. Kann auf eine Dekomprimierung verzichtet werden, weil die Dateien auch so benutzt werden können, verringert das womöglich die Qualität der Datei, so wie es z.B. mit komprimierten Bildern der Fall ist. Es gibt aber auch die Möglichkeit, die zu übermittelnden Dateien geschickt zu strukturieren bzw. sie in geeigneten Formaten zu schicken. Beispielsweise könnte eine kartenanzeigende Applikation nur den Bereich der Karte schicken, die der Nutzer gerade betrachten möchte. Somit werden die Kacheln immer nachgeladen, wenn der Nutzer sich andere Bereiche der Karte anschauen möchte. Das ist zwar besser als die ganze Karte in einer Bilddatei zu schicken, aber bringt das Problem mit, dass Wartezeiten entstehen, wenn Kacheln nachgeladen werden müssen.
Gerade in solchen Szenarien ist es besser, die geographischen Informationen in einer sogenannten *Vektordatei* zu bewahren. Eine Vektordatei ist eine textbasierte Datei, die die Koordinaten und weitere Informationen der Objekte enthält. Mit diesen Informationen ist es möglich, eine korrekte Repräsentation dieser Daten zu visualisieren. Solche *Vektordaten* sind den Bildern, die die Karte repräsentieren (sogenannte *Rasterdaten*) vorzuziehen, wenn die Größe der Dateien eine Rolle spielt, da korrespondierenden *Vektordaten* kleiner sind als die *Rasterdaten* [Fling, 2009].

**Motivation** :
Im letzten Abschnitt wurde eins von vielen möglichen Szenarien beschrieben, in denen Vektordaten den Rasterdaten vorzuziehen sind. Je nachdem von welchen Quellen die Geodaten stammen, sind sie in Form von Vektordaten oder von Rasterdaten vorhanden. Der Anbieter dieser Daten kann auch beide Formen anbieten. Um Vektor- und Rasterdaten anzubieten wurden internationale Standards eingeführt, damit es leichter wird, die gewünschten Daten zu finden, sie zu bekommen und einzusetzen. Ein Standard um Rasterdaten anzubieten ist der *Web Map Service* (WMS)[1]. Für den Fall, dass die benötigten Geodaten nur in Form von Rasterdaten liegen, kann es notwendig werden diese in Vektordaten umzuwandeln. Mithilfe von *Geoinformationssystemen* (GIS) ist es möglich die Rasterdaten einzulesen und dann manuell in Vektordaten zu konvertieren. Das ist aber ein langer und mühsamer Prozess und bietet sich daher an, automatisiert zu werden. Es gibt Datenanbieter, wie z.B. die Berliner Senatsverwaltung, die sehr viel Kartenmaterial zu verschiedenen Themen für Berlin hat. Dort ist es möglich, die visualisierten Daten zu betrachten. Im Hintergrund läuft ein WMS-Server (siehe Kapitel 2.2), der die geforderten Rasterdaten anbietet. Es wäre aus bereits genannten Gründen vorteilhaft, diese Daten ebenfalls in Form von Vektordaten vorliegen zu haben. Es sind aber mehrere Hundert solcher Rasterdatensätze vorhanden und diese manuell in Vektordaten zu konvertieren wäre ein sehr zeit- und kostenaufwändiges Unterfangen. Aus diesem Grund wurde im Rahmen dieser Bachelorarbeit ein Umwandlungsdienst erstellt, der diese Rasterdaten in Vektordaten umwandeln kann. Um die Ergebnisse betrachten zu können wurde ebenfalls eine webbasierte Visualisierung ermöglicht.

**Gliederung** :
Die vorliegende Bachelorarbeit ist wie folgt gegliedert: Im zweiten Kapitel wird der Begriff *Geoinformationssystem* näher erläutert und es werden einige Standards in dieser Disziplin gezeigt. Eine der großen Organisationen, die in diesem Bereich tätig ist, ist das *Open Geospatial Consortium*. Es veröffentlichte Standards zur Findung und zum Austausch von Geodaten, wie z.B. den *Web Map Service*, den *Web Feature Service* und den *Web Coverage Service*. Besonderer Wert wird auf den *Web Map Service* gelegt, da er benutzt wird, um Rasterdaten auszugeben. Kapitel 3 beschäftigt sich zunächst mit geographischen Themen, wie z.B. der *Koordinatentransformation*. Es gibt verschiedene Koordinatensysteme mit denen man die Position von Orten der Erde bestimmen kann. Diese unterscheiden sich bei ihrer Genauigkeit. Es gibt lokale Koordinatensysteme, wie das *EPSG:3068*, die für den deutschen Raum beson-

[1] http://www.opengeospatial.org/standards/wms - Zugriff: 20.11.2014

ders genau sind und globale Koordinatensysteme, wie das *EPSG:4326*, die überall präzise, aber nie so präzise wie lokale Koordinatensysteme an den spezifischen Orten sind. Außerdem wird in Kapitel 3 das Vektor- und Rastermodell vorgestellt, über die bereits berichtet wurde. In Kapitel 4 werden Algorithmen vorgestellt, die bei der Implementierung des Umwandlungsdienstes zum Einsatz kommen. Es werden zuerst Algorithmen vorgestellt, die die Daten bearbeiten, und dann welche, die diese Daten korrigieren. Im fünften Kapitel werden die Ausgangsdaten näher erläutert, die von dem Umwandlungsdienst, welcher im Rahmen dieser Bachelorarbeit entwickelt wurde, konvertiert wurden. Anschließend wird der Umwandlungsdienst vorgestellt und mit Hilfe einer Evaluation die Stärken und Schwächen ermittelt, um in der darauffolgenden Diskussion zu erörtern, inwiefern der Umwandlungsdienst eine bessere Möglichkeit bietet, die Rasterdaten in Vektordaten zu konvertieren. Kapitel 6, welcher dazu dient, die Ergebnisse der Bachelorarbeit zu diskutieren und einen Ausblick auf die potenziellen Erweiterungsmöglichkeiten des Umwandlungsdienstes gibt, schließt die Arbeit ab.

# 2 Standardisierung in Geoinformationssystemen

In diesem Kapitel wird erläutert, was man unter Geoinformationssystemen (GIS) versteht und wozu sie gebraucht werden. Außerdem wird eine Organisation vorgestellt, die viele Standards in GIS gesetzt hat, und zum Schluss werden einige dieser Standards vorgestellt.
Unter einem GIS versteht man ein integriertes Softwaresystem, das die Prozesse Geodatenerfassung, Geodatenverwaltung, Analyse Visualisierung/Kartographie sowie ihre Wechselwirkung unterstützt. Somit kommen Geoinformationssysteme für viele Fälle in Frage. Folgende Fragen muss ein GIS beantworten können [Heywood et al., 2006]:

- Frage nach einem Ort (*Bsp.: Wo ist die nächste Bücherei?*)
- Frage nach einem Muster (*Bsp.: In welchen Stadtteilen leben viele Studenten?*)
- Frage nach Trends (*Bsp.: Wie wird der Einzelhandel innerhalb der Stadt durch Einkaufszentren, die sich außerhalb der Stadt befinden, beeinflusst?*)
- Frage mit Bedingungen (*Bsp.: Wo befinden sich in einem Radius von 5 km um einen Hauptbahnhof herum über 100.000 potenzielle Kunden?*)
- Frage mit Implikationen (*Bsp.: Wie wird die Verkehrslage dieser Straße beeinflusst, wenn wir an jenem Ort einen Vergnügungspark eröffnen?*)

Solche Fragen müssen oft im privaten, staatlichen sowie akademischen Sektor beantwortet werden. Infolgedessen steigt die Anzahl der GIS-Experten, aber auch unprofessionelle Nutzer [Decker, 2001].
Aus diesem Grund sind Geoinformationssysteme sehr gefragt [DeMers, 2009] und es gibt eine breite Auswahl von sowohl kommerziellen als auch Open-Source-Lösungen. Zu den kommerziellen GIS gehören Map3D[2], und ArcGIS[3] und die verbreitetsten Open-Source-GIS: GrassGIS[4] und QGIS[5].
Es ist vorteilhaft Standards im Bereich der Geoinformationssysteme einzuführen [Percivall, 2010]. (siehe Kapitel 3.1.1). Zu diesem Anlass haben sich Vertreter aus Wirtschaft, Staat und Forschung zusammen getan, um das Open Geospatial Consortium Inc. (OGC) zu gründen. In den nächsten Unterkapiteln werden das OGC sowie einige seiner Standards vorgestellt.

[2] http://www.autodesk.de/products/autocad-map-3d/overview/ - Zugriff: 17.11.2014
[3] https://www.arcgis.com/ - Zugriff: 17.11.2014
[4] http://grass.osgeo.org/ - Zugriff: 17.11.2014
[5] http://www.qgis.org/ - Zugriff: 17.11.2014

## 2.1 Open Geospatial Consortium

Das Open Geospatial Consortium (OGC[6]) ist „eine Vereinigung mehrerer rechtlich und wirtschaftlich selbstständig bleibender Unternehmen zur Durchführung eines vereinbarten Geschäftszweckes.“ [Woll, 1992]. Es wurde im Jahre 1994 gegründet und hat inzwischen mehr als 500 Organisationen aus Wirtschaft, Staat und Wissenschaft. Eines der Hauptmerkmale des OGC ist es offen zu sein und die Programme Open-Source zu betreiben. Das OGC verfolgt das Ziel die Entwicklung und Nutzung von internationalen Standards zu fördern [7]. Dadurch wird die Interoperabilität, Portabilität und Wartbarkeit von Programmen erhöht [Richard Groot, 2000].
Neben dem OGC gibt es noch weitere Organisationen, die internationale Standards für Geoinformationssysteme einführen, um die drei genannten Vorteile zu erhalten, wie z.B. International Organization For Standardization (ISO)[8] und das American National Standards Institute (ANSI)[9].

## 2.2 OGC-Standards

OGC-Standards sind “technische Dokumente, die Schnittstellen, Kodierungen, Profile, Anwendungsschemata, sowie Best Practices ausführlich beschreiben.”[10] Die Standards liefern eindeutige Spezifikationen, um die Implementierung von Interfaces zu ermöglichen, die eine standardisierte Form der Geodatenverarbeitung ermöglichen [Di, 2008]. Durch bekannte Mitglieder wie Google, Microsoft und NASA hat sich das OGC zu einer renommierten Organisation entwickelt, sodass Vorschläge des OGC in der Allgemeinheit schnell übernommen werden. [Kralidis, 2008].
Die in diesem Kapitel vorgestellten Standards sind Komponenten der OGC Web Services Architektur(OWS)[11]. Bei der OWS handelt es sich um eine service-orientierte Webdienstarchitektur, welche in der Informatik eine verbreitete Technik ist:

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

[6] http://www.opengeospatial.org/ - Zugriff: 18.11.2014
[7] http://www.opengeospatial.org/ogc/vision - Zugriff: 01.12.2014
[8] http://www.iso.org/iso/home.html - Zugriff: 01.12.2014
[9] http://www.ansi.org/ - Zugriff: 01.12.2014
[10] http://www.opengeospatial.org/standards - Zugriff: 20.11.2014
[11] http://www.opengeospatial.org/standards/owc - Zugriff: 02.12.2014

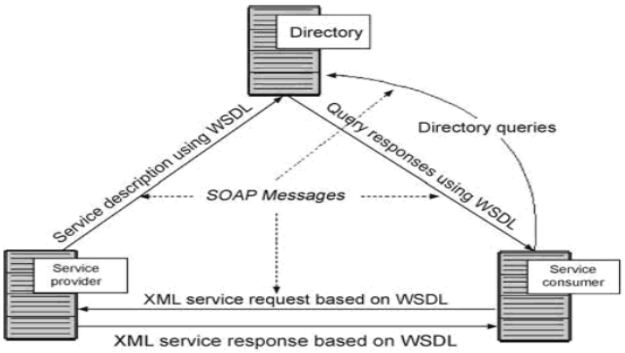


**Abbildung 1:** *Service-orientierte Architektur eines Webdienstes [Quelle: Mapping Hacks by Schuyler Erle, Rich Gibson and Jo Walshs (2005)]*

*XML serialization in conjunction with other Web-related standards."*[12]

In Abbildung 1 ist eine schematische Darstellung eines Webdienstes mit einer service-orientierten Architektur zu sehen. Der *Service Provider* kann z.B. ein Web Map Service sein. Er nimmt Anfragen bezüglich von Geodaten an und beantwortet sie je nach Dienst mit einem Dokument oder einer Bilddatei. Es ist ebenfalls die Aufgabe eines *Service Providers* sich in einem Katalog einzutragen, damit es von *Service Consumern* benutzt werden kann. Die *Directory* dient als Katalog, um die jeweiligen Server, ihre Adressen in Form von URL und ihre Dienste zu finden, um sie entsprechend ansprechen zu können. Der *Service Consumer* ist der Nutzer dieser Dienste. Er schaut in der *Directory* nach vorhandenen Servern und schickt Anfragen an einen Server seiner Wahl. Die Kommunikation erfolgt über das HTTP. Dabei ist sowohl die GET-Methode mit einem Key-Value möglich als auch die POST-Methode mit einem XML-Dokument [Percivall, 2008b].

**Catalog Service for the Web** Der Catalog Service for the Web (CSW) ist ein Webdienst, der in der oben beschriebenen service-orientierten Architektur die Rolle der *Directory* übernimmt. Laut der CSW-Spezifikation muss ein CSW aus zwei Komponenten bestehen [Yue et al., 2010]. Die erste Komponente ist ein abstraktes Informationsmodell, welches aus einem Schema für den Katalog und einer *Anfragesprache* für Kataloge besteht. Die zweite Komponente ist ein *HTTP Verbindungsprotokoll*, das dem Zweck dient über das HTTP erreichbar zu sein.

**Web Map Service** Der Web Map Service (WMS) ist ein von OGC im Jahre 2000 veröffentlichter Standard. Er war die erste große Publikation des OGC und hat eine

[12] http://www.w3.org/TR/ws-gloss/#webservice - Zugriff: 20.11.2014

große Rolle dabei gespielt, den Bekanntheitsgrad des OGC zu erhöhen [Kralidis, 2008]. Beim WMS handelt es sich um ein OWS. Der WMS ist ein Webdienst, der in der Lage ist, Kacheln in Form von Rasterdaten an den Klienten zu schicken.
Um die Dienste eines WMS-Servers zu beanspruchen, muss ein Klient zunächst wissen, welche WMS-Server es gibt und unter welcher Adresse diese erreicht werden können. Zu diesem Zweck hat das OGC das CSW entwickelt. Hat ein Klient einen Server gefunden und weiß, wie er ihn ansprechen muss, kann er ihm Anfragen schicken, indem er die HTTP-GET-Methode verwendet und an die Serveradresse ein Fragezeichen, gefolgt von einer Reihe von Parametern, anhängt und die Anfrage sendet. Diese Parameter sind Versionsnummer, Operation, von der Operation abhängige weitere Parameter. Schematisch sieht eine Anfrage des Klienten wie folgt aus:

```
<Serveradresse>?&version=<version>&request=<Operation>&
```

Die Reihenfolge der Parameter hinter dem Fragezeichen ist beliebig. Der WMS-Server nimmt die Anfrage(n) eines Klienten bezüglich der Koordinaten von Bounding Boxes, Style, Format und Auflösung entgegen und sendet falls möglich die entsprechenden Rasterdaten als Bilder zurück. Eine Bounding Box ist ein Rechteck auf einer Karte, das aus vier Eckkoordinaten besteht.
Der Klient erhält ein Bild und keine Rohdaten. Er kann die Bilder zwar anzeigen, aber es ist ihm nicht möglich weitere Informationen aus dem Bild zu erhalten, ohne weitere Bildanalysetechniken zu verwenden. Der WMS ist überwiegend für den visuellen Gebrauch geeignet, da die Antwort aus Bildern besteht, die angezeigt werden können [Michaelis and Ames, 2008]. Diese Bilder werden aus Rohdaten gerendert, was bei großen Bildern für viel Serverlast sorgen kann.
In der WMS-Spezifikation [wms, ] sind zwei Operationen spezifiziert, die alle WMS-Server zur Ausführung anbieten müssen.
Im Folgenden wird auf die drei Operationen eingegangen und Beispiele vorgestellt.
**GetCapabilities**: Diese Anfrage beantwortet der WMS-Server mit einem XML-Dokument, das Metadaten über die unterstützten Ebenen und Operationen enthält. In Anhang 1 ist eine Beispielantwort auf eine GetCapabilities-Anfrage enthalten.
Im Service-Element gibt es Informationen über den Dienst wie Name des Dienstes, Betreiber usw. Die vorhandenen Ebenen und ausführbaren Operationen sind im Capability-Element enthalten. Die ausführbaren Operationen sind an den Elementen **<GetCapabilities>** und **<GetMap>**zu erkennen. Nach den Operationen kommen die anforderbaren Ebenen, die in den Layer-Elementen näher spezifiziert sind.

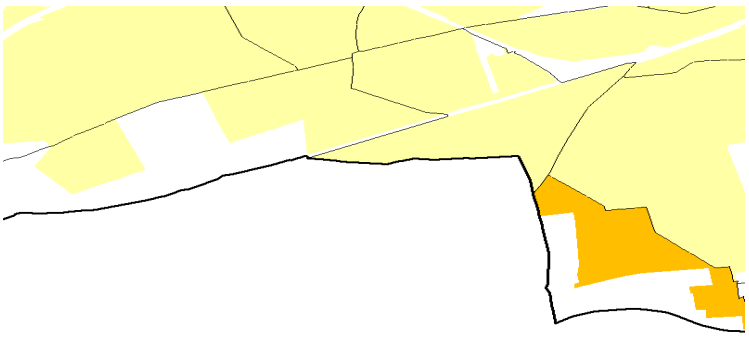

**Abbildung 2:** *Antwort eines WMS-Servers auf eine GetMap-Anfrage [Quelle: [13]]*

**GetMap**: Die GetMap-Anfrage gibt die Kachel aus, die durch den Klienten angefragt wurde. Dabei muss der Klient neben der Versionsnummer und der GetMap-Anfrage auch das Koordinatensystem (siehe Kapitel 3.1), die Bounding Box, die gewünschten Ebenen sowie Format und Auflösung der Bilder angeben. Sind die Angaben korrekt, rendert der Server das Bild und schickt es dem Klienten. Sonst wird eine Fehlermeldung gesendet.
Anfrage: http://fbinter.stadt-berlin.de/fb/wms/senstadt/MsozSarblos252006LOR?VERSION=1.3.0&REQUEST=GetMap&CRS=EPSG:3068&BBOX=10750,8030,15700,12120&WIDTH=800&HEIGHT=600&LAYERS=0&STYLES=&FORMAT=image/png (Zugriff: 03.12.2014)
Antwort: Siehe Abbildung 2
In der Anfrage sind die verschiedenen Key-Value-Paare aufgelistet, die zu dem Ausgabebild führen. Die verschiedenen Key-Value-Paare sind mit einem &-Zeichen getrennt. Die Reihenfolge spielt keine Rolle.

**Web Feature Service** :
Der Web Feature Service[14] (WFS) ist ein vom OGC im Jahre 2002 veröffentlichter Standard. Der WFS ist ein Webdienst, der Features anbietet. Wie der WMS weist auch der WFS eine OWS Architektur auf. Da die Geodaten als Vektordaten (siehe Kapitel 3.2) vorliegen, kann der Klient aus diesen Daten eine Karte erstellen oder seine eigene Karte ergänzen. Die in der WFS Spezifikation als Schnittstelle für die Anfragen definierten Operationen sind *GetCapabilities*, um die Informationen über

[13] http://fbinter.stadt-berlin.de/fb/wms/senstadt/MsozSarblos252006LOR?VERSION=1.3.0&REQUEST=GetMap&CRS=EPSG:3068&BBOX=10750,8030,15700,12120&WIDTH=800&HEIGHT=600&LAYERS=0&STYLES=&FORMAT=image/png - letzter Zugriff: 03.12.2014
[14] OpenGIS Web Feature Service (WFS) Implementation Specification in www.opengeospatial.org/standards/wfs - Zugriff: 20.11.2014

den WFS-Server zu erhalten, DescribeFeatureType um Informationen über einen bestimmten Punkt auf der Karte zu erhalten und GetFeature um ein Feature anzufordern [Vretanos, 2005].

**Web Coverage Service** :
Der Web Coverage Service (WCS) ist ein vom OGC im Jahre 2003 veröffentlichter und 'Coverages' liefernder Standard. Bei einem 'Coverage' handelt es sich um einen Datentyp, der kontinuierliche Abweichungen eines Werts in einem Raum abbilden kann, wie z.B. die Bodenfeuchtigkeit in einem Feld [Open Geospatial Consortium, 2007].
Im Web Coverage Service 2.0 Interface Standard - Core: Corrigendum [Open Geospatial Consortium, 2010] sind Operationen und Strukturen für die Antworten definiert, die alle WCS-Dienste einhalten müssen. Damit ein WCS als solcher bezeichnet werden kann, muss es mindestens den WCS Kern implementieren. Im Kern hat man darauf geachtet, Platz für Erweiterungen zu lassen.
Im Kern sind 3 fundamentale Operationen beschrieben: *GetCapabilities* um Informationen über den WCS-Server zu erhalten, *DescribeCoverage* um Informationen über die bereitgestellten Coverages zur erhalten und *GetCoverage* um eine Coverage anzufordern. Es ist möglich den WCS mit anderen Standards und Spezifikationen zu kombinieren, um verschiedene Ergebnisse zu erhalten. So kann ein WCS benutzt werden, um eine Visualisierung der Daten im Internet-Browser zu ermöglichen [Lee et al., 2005].

**WMS, WFS und WCS - Ein Vergleich** :
Nachdem die drei Dienste WMS, WFS und WCS vorgestellt wurden, soll hier nochmal ein Vergleich zwischen ihnen gemacht werden, um Gemeinsamkeiten und Unterschiede zu erkennen sowie einen besseren Überblick zu bekommen.
Gemeinsamkeiten:

- **OpenGIS Web Services (OWS)**: Alle drei Dienste gehören zur Gruppe der OWS und sind somit Dienstanbieter in einer service-orientierten Webdienstarchitektur.
- **Geodatenanbieter**: Die drei Dienste bieten den Klienten Geodaten in verschiedenen Formaten an.
- **Interoperabilität**: Durch die Verwendung von XML und GML wird ein hoher Grad an Interoperabilität erreicht [Michaelis and Ames, 2008]

- **Operationen**: *GetCapabilities*, um die ausführbaren Operationen sowie die Daten, die die Dienste anbieten, zu erhalten; *DescribeLayer, FeatureType,Coverage* um nähere Informationen über eine(n) bestimmte(n) Ebene, FeatureType, Coverage zu erhalten; *GetMap,Feature,Coverage* um die Geodaten, wie in der Anfrage spezifiziert wurde, zu erhalten.

Wie man an den Gemeinsamkeiten erkennt, sind die unterschiedlichen Dienste von ihrem groben Aufbau und ihrer groben Struktur sowie Aufgaben sehr ähnlich. Die Unterschiede jedes einzelnen zu den anderen beiden werden deutlicher, je detaillierter man die Dienste betrachtet, wie Tabelle 1 verdeutlicht.

**Tabelle 1:** *WMS, WFS und WCS Überblick [Percivall, 2008a]*

| | Anfrage* | Rückgabetyp | Rohdaten möglich? | Datensätze manipulierbar? |
|---|---|---|---|---|
| WMS | 1 | Bilder | nein | nein |
| WFS | 2 | Features | ja | ja |
| WCS | 3 | Coverages | ja | ja |

*Anfrage: Je höher die Zahl, desto komplexere Anfragen sind möglich*

In diesem Kapitel ist deutlich geworden, dass der WMS aufgrund seiner begrenzten Möglichkeit die Geodaten lediglich als Bilder auszugeben überwiegend für Nutzer, die nur an Kartenanwendungen interessiert sind, geeignet ist. Durch seine Einfachheit hat er einen großen Bekanntheitsgrad erlangt [Davis, 2007].
Sowohl der WFS als auch der WCS hingegen sind für Nutzer geeignet, die sich besser mit GIS auskennen, da für die Anfragen spezielle Kenntnisse erforderlich sind.In vielerlei Hinsicht ist der WFS oder der WCS vorzuziehen. Die Möglichkeit aus den Rohdaten, die diese Dienste anbieten, weitere Daten zu generieren oder sie mit anderen Daten zu kombinieren ist der Grund dafür.

# 3 Formate und Datentypen von Geodaten

Geodaten können auf viele verschiedene Arten generiert und digitalisiert werden. Es gibt einfach und schnell realisierbare Methoden, wie z.B. das Scannen einer gedruckten Karte [Decker, 2001]. Als Ergebnis entsteht ein Bild und somit ist die Karte nun als Rasterdatei (siehe Kapitel 3.2) vorhanden. Man kann das Bild anschließend am Computer georeferenzieren, indem man z.B. einigen Pixeln eine Koordinate zuordnet. Anschließend kann die Rasterdatei in ein Geographisches Informationssystem (GIS) eingebettet werden [DeMers, 2009].
Eine etwas aufwändigere, aber genauere Art Karten zu generieren ist es mit GPS-Empfängern das gewünschte Gebiet vor Ort aufzuzeichnen. Dafür ist es notwendig mit diesen Geräten die Koordinaten der Eckpunkte für das Objekt zu ermitteln, welches geokodiert werden soll. Als Ergebnis erhält man Dateien, die mehrere Koordinaten enthalten. Je nach Gerät ist es möglich weitere Attribute zu setzen. Da es sich beim Ergebnis um eine Textdatei mit Koordinaten handelt, spricht man dabei von einer Vektordatei (siehe Kapitel 3.2).
In GIS wird hauptsächlich zwischen diesen beiden Formaten unterschieden. Beide Formate haben ihre Vor- und Nachteile, die in den folgenden Unterkapiteln erläutert werden, und je nach Verwendungszweck sind Geodaten in der einen oder anderen Form vorzuziehen.
Eine weitere Methode, um Kartenmaterial zu erhalten, ist es mit Satellitenaufnahmen zu arbeiten. Weil diese Methode die (kosten-)aufwändigste Methode ist, wird sie oft von Staaten oder großen Firmen durchgeführt. In den folgenden Unterkapiteln wird es unter anderem um Koordinatensysteme gehen. Diese sind nämlich wichtig, um Geodaten zu generieren und darstellen zu können. Des weiteren wird auf die zwei Arten von Formaten bzw. Modellen (Rastermodell und Vektormodell) eingegangen, um Unterschiede, Vor- und Nachteile, sowie die Anwendungsgebiete zu erhellen. Danach werden zwei verschiedene Formate für Vektordaten, nämlich GeoJSON und GML, kurz samt ihrer Vor- und Nachteile vorgestellt.

## 3.1 Koordinatensysteme und -transformationen

Um Geodaten zu visualisieren ist ein System nötig, welches die Beschreibung der Lage von Objekten auf der Erde ermöglicht. Bereits in der Antike hat der griechische Gelehrte Eratosthenes von Kyrene etwa 260 v. Chr. ein Gitter als Koordinatensystem benutzt, um die Position von Objekten zu bestimmen. Diese Idee hat sich bis heute gehalten, denn immer noch beruhen die meisten geografischen Koordinatensysteme auf einem (Kartesisches) Koordinatensystem. Allerdings wird ein geeignetes Koordinatensystem als Referenz benötigt. Es muss einen Bezug zur Erde haben, wie z.B. die Lage des Koordinatenursprungs und die Richtung der Achsen. Dieses Koordinatensystem wird als ein Koordinatenreferenzsystem bezeichnet.
Wäre die Erde eine flache Ebene, könnte ein einfaches Kartesisches Koordinatensystem benutzt werden. Dazu wäre lediglich die Festlegung auf einen Ort als Koordinatenursprung und die Einheiten der Achsen nötig. Somit hätte jeder Ort seine eigenen Koordinaten und wäre exakt bestimmbar. Da die Erde aber kugelförmig ist, ist dies nicht ohne weitere Überlegungen möglich. Das Problem ist die Darstellung einer dreidimensionalen Erdoberfläche auf einem zweidimensionalem Bezugssystem [Krygier and Wood, 2011].

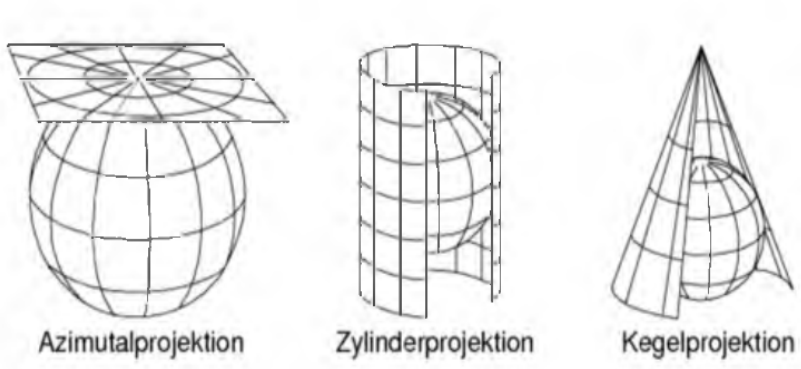


**Abbildung 3:** *Klassifikation der Kartennetzentwürfe nach der Art der Abbildungsflachen [Quelle: Geoinformatik in Theorie und Praxis 2. Auflage N. de Lange Abbildung 5.10]*

Wie in Abbildung 3 dargestellt, gibt es drei Arten der Kartennetzentwürfe: Azimutal-, Kegel- und Zylinderprojektion. **Azimutalprojektionen** sind insbesondere dafür nützlich Polregionen darzustellen, da es bei Anwendung der anderen Projektionen zu Verzerrungen kommt. Die Vorteile von **Kegelprojektionen** sind eine mögliche Längen- und Winkeltreue, sowie eine geringe Verzerrung an der Stelle, wo der Kegel die Erde berührt. Je größer die Distanz zu diesem Punkt wird, desto größer ist jedoch die Verzerrung. Bei den meisten **Zylinderprojektionen** hat der Zylinder

dieselbe Ausrichtung wie die Erdachse (siehe Abbildung 3). Die resultierende Karte zeigt eine stärker werdende Verzerrung an den Polregionen. Dadurch entsteht bei solchen Karten der falsche Eindruck, dass Landmassen in den Polnähen (wie z.B. Grönland) größer sind als jene, die näher am Äquator liegen (z.B. Afrika) [de Lange, 2007].
Die Erde ist jedoch keine perfekte Kugel, sondern ein Körper, das einem Rotationsellipsoid nahe kommt. Sie ist an den Polen abgeflacht und weist viele "Dellen" auf [Feeman, 2008]. Diese Tatsache macht es unmöglich ein fehlerfreies globales Koordinatensystem zu entwerfen. Globale Koordinatensysteme werden bei höherer Auflösung immer ungenauer. Deshalb haben verschiedene Staaten und Staatenbunde verschiedene lokale Koordinatensysteme entwickelt, die jeweils eine Ellipse als Basis haben. Diese Ellipse ist je nach Krümmung der Erdoberfläche in jenem Gebiet geformt. So wurden im Laufe der Zeit durch verschiedene Akteure sehr viele verschiedene Koordinatensysteme entworfen. Daraus entsteht das Problem manchmal die Koordinaten zwischen den Systemen umzuwandeln, also Koordinatentransformationen durchzuführen. Um Koordinatentransformationen durchführen zu können, sind je nach Variante der Transformation verschiedene Parameter der Koordinatensysteme notwendig. Diese Parameter werden zusammen mit einer ID für die Koordinatenreferenzsystem vom European Petroleum Survey Group Geodesy (EPSG)[15] verwaltet. Einige relevante Beispiele:

- **EPSG:4326** : Auch bekannt als WGS84, projiziert die ganze Welt. Hat sich in vielen Gebieten als Standardprojektion durchgesetzt.
- **EPSG:3857** : Auch bekannt als Web Mercator, projiziert die ganze Welt. Wird überwiegend auf Webbasierten Karten benutzt (z.B. Google Maps)
- **EPSG:3068** : Auch bekannt als DHDN / Soldner Berlin, projiziert Berlin und Umgebung.

## 3.2 Vektor- und Rastermodell

Im Kontext von GIS ist ein Datenmodell ein mathematisches Konstrukt, um geographische Objekte zu repräsentieren [Hoel, 2008]. Dabei erfolgt die Darstellung von geographischen Objekten im *Vektor*- oder im *Rastermodell*, die als Punkt (z.B. eine Markierung für eine Tankstelle), als (Poly-)Linie (z.B. eine Straße oder eine Grenze) und ein Polygon (z.B. ein See) gezeigt werden. Dem *Rastermodell* liegt eine quadratische Masche zugrunde, in der ein Pixel die kleinste Einheit bildet. Alle Pixel

[15] http://www.epsg.org/ - Zugriff: 04.01.2015

sind gleich groß und können eindeutig identifiziert werden, da dem Rastermodell ein kartesisches Koordinatensystem zugrunde liegt. Geographische Objekte werden durch benachbarte Pixel einer Farbe markiert. Dadurch kann ein Pixel einen Punkt, mehrere Pixel eine Linie oder einen Polygon darstellen.
Beim *Vektormodell* geschieht die Repräsentation der Objekte mittels Vektoren in einem kartesischen Koordinatensystem. Allerdings muss hier die Information, ob es sich um einen Punkt, eine Linie oder ein Polygon handelt, ergänzt werden. Die Tabelle 2 stellt die Vor- und Nachteile des Vektor- und Rastermodells gegenüber, wobei die Vorteile des einen die Nachteile des anderen sind und umgekehrt. Die Tabelle verdeutlicht, dass kein Modell allgemein besser als das andere ist, da für jede Situation entweder das Vektor- oder das Rastermodell zu bevorzugen ist [de Lange, 2007].

**Tabelle 2:** *Vergleich zwischen dem Vektor- und Rastermodell*

| **Vorteile des Vektormodells** | **Vorteile des Rastermodells** |
|---|---|
| • hohe geometrische Genauigkeit<br>• eindeutige Objektbeschreibung<br>• geringe Datenmengen<br>• graphische Präsentation ist traditionellen Karten ähnlicher<br>• einfache Koordinatentransformation | • einfache Datenstrukturen<br>• kompatibel mit Fernerkundungs- und Scannerdaten<br>• einfache logische und algebraische Operationen |

Da im Rahmen dieser Bachelorarbeit ein Dienst entwickelt wird, welcher Rasterdaten in Vektordaten umwandelt, muss ein Format eines Vektormodells ausgewählt werden, das am Ende ausgegeben wird. Es gibt eine Vielzahl von Vektorformaten, jedoch haben sich überwiegend das Geographic Markup Language (GML)[16] und GeoJSON[17] durchgesetzt [Kralidis, 2008] [Morris, 2010]. Beide Formate sind textbasiert und dazu gedacht mit ihnen geographische Informationen zu übermitteln. Dies geschieht mittels Koordinaten (ein Zahlenpaar). Dementsprechend können beide die Grunddatentypen für geographische Objekte bilden, die da wären: Punkt, (Poly-)Linie, Polygon. Außerdem ist es bei beiden möglich das zugrunde liegende Koordinatensystem einzubauen. GeoJSON ist eine Formalisierung von JSON[18] und

[16] http://www.opengeospatial.org/standards/gml - Zugriff: 24.02.2015
[17] http://geojson.org/ - Zugriff: 24.02.2015
[18] http://json.org/ - Zugriff: 24.02.2015

GML ist eine Grammatik im XML-Schema und somit bauen sowohl GeoJSON, als auch GML auf bereits bekannten Formaten auf.

### 3.2.1 Geographic Markup Language

Das Geographic Markup Language (GML) ist ein vom OGC entwickelter Standard zur Speicherung von georeferenzierten Daten. Mittlerweile hat es drei Versionen (GML 1, 2 und 3) von 1999 bis 2003 entwickelt. Mit jeder Version wurden neue Funktionalitäten hinzugefügt, sodass GML 3 z.B. auch verschiedene Zeitstempel der Geodaten berücksichtigen kann. Die aktuelle Version hat über 400 Seiten [Open Geospatial Consortium, 2012], was die Mächtigkeit des GML erahnen lässt. Zwei wichtige Komponenten von GML sind die Elemente Feature und FeatureCollection. Diese stellen geographische Objekte dar, wie z.B. Flüsse, Straßen und Wälder. Ebenfalls von großer Bedeutung sind Geometrie-Objekte, wie z.B. *gml:Point, gml:Curve, gml:LineString, gml:_Surface, gml:Polygon, gml:_Ring, gml:Circle*, und *gml:LinearRing*, welche z.B. die Grenze eines Bundeslandes repräsen-tieren können [gml, 2008]. *gml:coordinates* enthalten die Koordinaten. Ein Beispiel für Geodaten im GML-Format:

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <gml:FeatureCollection
3    xmlns:xsi="http://www.w3.org/2003/XMLSchema-instance"
4    xmlns:gml="http://www.opengis.net/gml">
5    <gml:featureMember>
6      <gml:location>
7        <gml:Point srsName="urn:ogc:def:crs:WGN84">
8          <gml:coordinates>52.5243, 13.4105</gml:pos>
9          <gml:LocationString>Berlin</gml:LocationString>
10       </gml:Point>
11     </gml:location>
12   </gml:featureMember>
13 </gml:FeatureCollection>
```

Es können aber noch viele weitere Informationen übertragen werden, wie z.B. die Einwohnerzahl, die Fläche etc., und der Empfänger der Datei weiß dank den vordefinierten Tags exakt, mit welchen Informationen er es zu tun hat. Die Möglichkeit, die Features sehr detailliert beschreiben zu können, ist ebenfalls ein Nachteil, da die GML-Dokumente so sehr groß werden und dadurch unattraktiver für die Übertragung über das Internet werden [Moretz, 2008].

### 3.2.2 GeoJSON

GeoJSON ist ein offener Standard, der von einer freien Gruppe von Entwicklern entwickelt wurde. Die GeoJSON Format Spezifikation[19] wurde im Jahre 2008 veröffent-

[19] http://geojson.org/geojson-spec.html - Zugriff: 24.02.2015

licht. In einem GeoJSON können geometrische Objekte, Features und eine Sammlung von Features (FeatureCollections) enthalten sein. Folgende Typen können ein geometrisches Objekt sein: Punkte (Point, MultiPoint), (Poly-)Linien (LineString, MultiLineString), Polygone (Polygon,MultiPolygon) und eine Sammlung von verschiedenen geometrsichen Objekten (GeometryCollection). Features im GeoJSON enthalten ein geometrisches Objekt und zusätzliche Eigenschaften sowie eine Sammlung von Features (FeatureCollection).
Jede GeoJSON-Datei ist ein JSON-Objekt und somit auch eine JSON-Datei. Ein GeoJSON-Objekt beinhaltet mehrere Schlüssel-Wert Paare. Die Schlüssel sind als String und die Werte können ein String, eine Zahl, ein Objekt, eine Liste oder eins der Literale "true", "false" oder "null" sein [Murray, 2013]. Ein Nachteil von GeoJSON ist, dass die Objekte keine vordefinierten Eigenschaften (properties) haben. Für das Koordinatensystem kann also der Schlüssel "crs", "coordinateSystem" oder "spatialReferenceSystem" gewählt werden. Das ist ein Punkt, der die Interoperabilität schwächt. Ein Beispiel für Geodaten im GeoJSON-Format:

```
{
  "type": "FeatureCollection",
  "crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" } },
  "features": [
    { "type": "Feature",
      "properties": { "LocationString": "Berlin" },
      "geometry": { "type": "point", "coordinates": [52.5243, 13.4105] }
    }
  ]
}
```

**Fazit:** Für den Umwandlungsdienst ist GeoJSON als Ausgabeformat besser geeignet, da die gewonnenen Geodaten nicht sehr viele zusätzliche Informationen enthalten. Diese müssen der Legende, die extra eingeblendet wird, entnommen werden. Dank der Tags ist GML etwas einfacher zu lesen, wodurch das Debuggen erleichtert wird, was aber keine sehr große Rolle spielt, da die Geodaten nicht sehr kompliziert aufgebaut sind, sondern nur aus Polygonen bestehen. Außerdem sind die erstellten Vektordaten sehr groß, wodurch es zu Leistungsproblemen während der Visualisierung kommen kann. Diese Tatsache begünstigt ebenfalls die Wahl für das GeoJSON-Format.

# 4 Umwandlungsalgorithmen

Höhlenmalereien auf der ganzen Welt deuten darauf hin, dass schon vor 30.000 Jahren Menschen sich bemüht haben simple Karten zu erstellen. Auf diesen Karten waren die Wanderrouten von Tierherden markiert, die sich für die Jagd und den Verzehr eigneten (z.B. Waldelefanten) [Woodward and Harley, 2007]. Auch in den darauf folgenden Zeitaltern wurden Karten immer wichtiger und immer besser bzw. genauer. Ohne Karten wären Handelsrouten schwerer entstanden und somit auch Zivilisationen und Reiche, wie z.B. das Antike Ägypten, Mesopotamien usw. Die Erstellung von Karten war jedoch mit viel Aufwand verbunden und die Präzision der Karten war je nach Zeitalter besser oder schlechter. So entstanden im Laufe der Jahrtausende Karten auf Haus- und Höhlenwänden, Stein- und Tontafeln, Papyri, Pergamente und Papier. Dabei haben sich die Werkzeuge ebenfalls je nach Medium geändert [Edney, 1996]. In Abbildung 4 ist eine der ältesten Landkarten zu sehen. Sie zeigt einen Stadtplan aus der Vogelperspektive und den Berg Hasan Dag. 1987 waren Reißbretter, Reißnadeln, Stifte und Lineale die bevorzugten Werk-

**Abbildung 4:** *Die älteste Landkarte der Welt in Catalhöyük/heutige Türkei [Quelle: http://grenzwissenschaft-aktuell.blogspot.de/2014/01/geologen-bestatigen-alteste-landkarte.html - letzter Zugriff: 13.01.2015]*

zeuge, um Karten herzustellen. Innerhalb des darauffolgenden Jahrzehnts wurden diese Werkzeuge durch den Computer ersetzt [Krygier and Wood, 2011]. In dieser Zeit sind Programme entstanden, die sowohl die Erstellung von Karten, als auch das Digitalisieren von gedruckten Karten ermöglichen. Zur Freude vieler Kartographen wurden diese Programme immer besser und immer mehr mühsame Schritte konnten automatisiert werden. Die schnelle Entwicklung und Verbesserung dieser Program-

me liegt unter anderem daran, dass Geoinformationssysteme zahlreiche aus anderen Bereichen der Informatik und Mathematik stammende Algorithmen enthalten, um ihren Zweck zu erfüllen. So finden viele Algorithmen, die numerische Berechnungen mit geometrischen Körpern durchführen, Einzug in GIS. Beispielsweise kann mit diesen Algorithmen bestimmt werden, ob zwei Straßen eine Kreuzung bilden bzw. ob sich zwei Geraden schneiden. Eine weitere Aufgabe von GIS ist es die Daten zu visualisieren. Hierzu gibt es ebenfalls viele verschiedene Algorithmen, die sich je nach dem Format und der Größe des darzustellenden Objekts ändern. Durch die Verbreitung von Mobilgeräten und Webbasierten GIS-Diensten wird die Größe der zu verschickenden Daten immer wichtiger und somit steigt die Relevanz verschiedener Komprimieralgorithmen, um z.B. Kartenkacheln, aber auch textbasierte Geo- und Metadaten zu komprimieren. Auch in dieser Arbeit wurden einige Algorithmen benutzt, um aus den von WMS-Servern gelieferten Kartenbildern die Polygone zu extrahieren, um diese anschließend zu geokodieren, damit sie in textbasierten Formaten gespeichert werden können. Diese Algorithmen werden in den folgenden Kapiteln vorgestellt und näher betrachtet. Manchmal kommt es vor, dass sich für die Lösung eines Problems mehrere Algorithmen anbieten. In so einem Fall werden diese vorgestellt, implementiert und in der Evaluierung (siehe Kapitel 5) ausgewertet, um zu bestimmen, welcher sich besser eignet. Die Algorithmen können in drei Aufgabengebiete unterteilt werden: Zuerst werden Algorithmen verwendet, die die Polygone aus den Bildern **extrahieren** und serialisieren sollen. Anschließend führen die dafür vorgesehenen Algorithmen eine **Korrektur** der Fehler durch, die während der **Extraktion** entstanden sind. Zum Schluss sorgen Umwandlungsalgorithmen für eine **Transformation** der Koordinaten in Georeferenzierungskoordinaten um.

## 4.1 Extraktion

Die Extraktion der in den Rasterdaten befindlichen Polygone ist der wichtigste Aspekt des im Rahmen dieser Arbeit entstehenden Umwandlungsdienstes. In den folgenden zwei Abschnitten werden die Polygone zunächst von eventuellen Grenzen, die die Polygone derselben Farbe aufteilen, beseitigt. Danach geht es weiter mit der Serialisierung jener Polygone.

### 4.1.1 Bereinigung der Kacheln von Grenzen

Ein Web Map Service (WMS) (siehe Kapitel 2.2) ist in der Lage verschiedene Informationsquellen anzuzapfen, um die Anfragen, die an ihn gestellt werden, zu erfüllen [wms, ]. Dabei spielt es für den WMS-Server keine Rolle, ob die Daten der

verschiedenen Quellen zueinander passen. Damit ist nicht der technische Teil gemeint, sondern die Genauigkeit bzw. Qualität der Daten. Es liegt an dem Betreiber dafür zu sorgen, dass die Daten zueinander passen. Beispielsweise können in der einen Datenquelle die Straßendaten einer Stadt aus dem Jahr 1900 stammen sein und in der anderen Datenquelle die Gebäude derselben Stadt aus dem Jahr 2000. Wird nun in der Anfrage nach den Straßen und den Gebäuden gefragt, werden diese zwei Ebenen vom WMS-Server aufeinander gelegt, gerendert und an den Klienten geschickt. Mögliche Kollisionen zwischen Straßen und Gebäuden sind für den WMS-Server irrelevant.
Sind die Datensätze unsauber, kann es zu unerwünschten, zusätzlichen Darstellungen auf den Ergebniskacheln kommen. Ein Beispiel soll das verdeutlichen: Der unter der URL
http://fbinter.stadt-berlin.de/fb/wms/senstadt/MsozSarblos252006LOR?
erreichbare WMS-Server bietet Informationen über die Arbeitslosenverteilung der Bevölkerung unter 25 Jahren im Jahre 2006 in Berlin an. Dafür sind drei Ebenen vorhanden: Ebene 0 für Straßen, Ebene 1 für die Arbeitslosenzahl, Ebene 2 für Planungsräume. In Abbildung 5 ist ein Ausschnitt der Karte aus dem oben

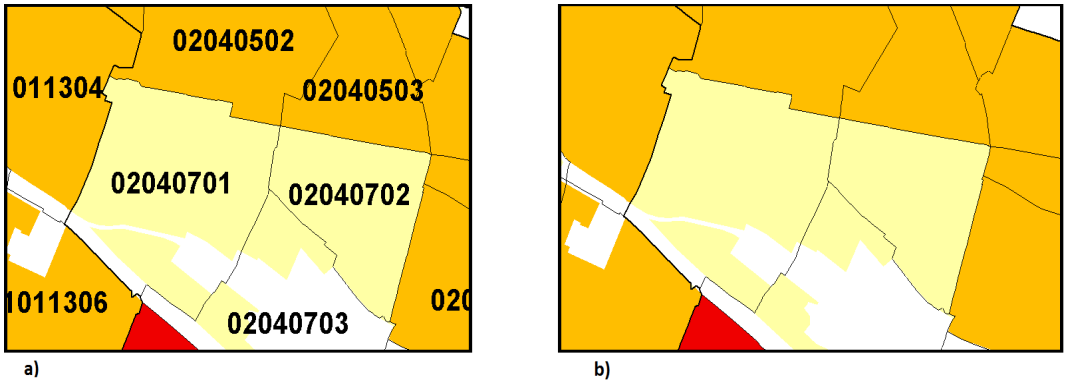


**Abbildung 5:** *a) Visualisierung der Arbeitslosenanzahl unter 25-jähriger in Berlin und Grenzen, sowie Nummern der Planungsräume b) Visualisierung der Arbeitslosenanzahl unter 25-jähriger in Berlin und Grenzen der Planungsräume [Quelle: http://fbinter.stadt-berlin.de/fb/index.jsp?loginkey=showMap&mapId=MsozSarblos252006LOR@senstadt (Zugriff:15.01.2015)]*

genannten WMS-Server zu sehen. In a) sind die Ebenen für die Arbeitslosenzahlen und die Planungsräume samt Grenzen und Nummern abgefragt worden. In b) hingegen wurde in der Anfrage lediglich die Ebene für die Arbeitslosen gefordert. Wie der Abbildung 5 zu entnehmen ist, werden unerwarteterweise die Grenzen der Planungsräume ebenfalls gerendert, obwohl dies laut Anfrage nicht verlangt war. Diese schwarzen Grenzen zerteilen die Polygone, erschweren und beschädigen die

Extraktion der Polygone, da sie mitten durch die farbigen Polygone verlaufen. Aus diesem Grund ist ein Algorithmus notwendig, der diese Grenzen nachträglich ändert.

Zunächst wurde ein Ansatz gewählt, der, wie es sich später herausstellen sollte, die schwarzen Linien zwar entfernt, dafür aber die Polygone stark verändert. Der Algorithmus sah vor die Grafik zeilenweise Pixel für Pixel von oben nach unten durchzugehen und die Farbe jedes schwarzen Pixels mit der Farbe des letzten nicht-schwarzen Pixels zu ersetzen.

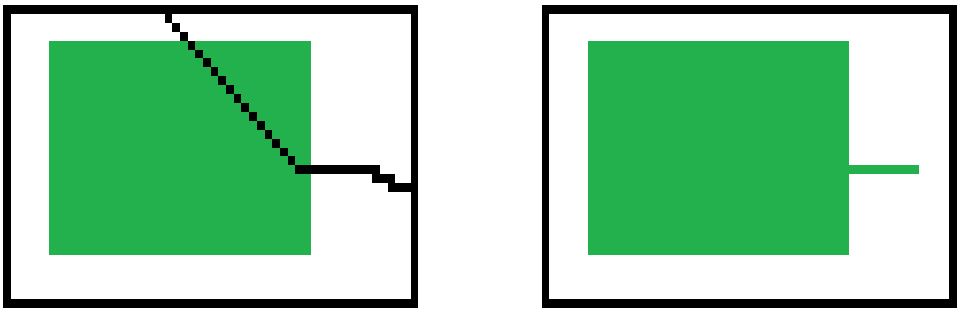

**Abbildung 6:** *Links: das ursprüngliche Bild, bevor die schwarzen Pixel entfernt wurden. Rechts: das veränderte Bild, nachdem die schwarzen Pixel entfernt wurden. [Quelle: Eigene Darstellung*

Abbildung 6 zeigt das Problem bei diesem Verfahren. Ein anderes Problem war die Behandlung des Auftretens eines schwarzen Pixels an erster Stelle, also an der linken oberen Ecke des Bildes. Wird dieser Fall nicht gesondert behandelt, werden alle folgenden schwarzen Pixel in derselben Reihe schwarz bleiben.

Der zweite Ansatz geht anders vor, um die Probleme, die beim ersten Ansatz auftreten, zu verhindern. Die Gemeinsamkeit besteht darin, dass beim Fund eines schwarzen Pixels wieder die Farbe der anderen Pixel betrachtet wird. Der Algorithmus bildet ein Quadrat, welches sieben Pixel lang und sieben Pixel hoch ist. Das Quadrat wird so positioniert, dass der schwarze Pixel in der Mitte ist. Anschließend werden die Farben aller Pixel betrachtet, die in diesem Quadrat sind und die Häufigkeit der Farben gezählt. Die Farbe, die am häufigsten vorkam wird für das schwarze Pixel übernommen.

### 4.1.2 Serialisierung der Polygone

Der wichtigste Schritt, um Rasterdaten in Vektordaten umwandeln zu können, ist es die Polygone zu serialisieren. Insbesondere sind die Eckpunkte wichtig, die ein Poly-

gon definieren, und die Reihenfolge der Eckpunkte, um eine korrekte Verbindung der Polygone darstellen zu können. Um dies zu erreichen, muss das Bild Pixel für Pixel durchlaufen werden und auf die Farbwerte (damit sind die RGBA-Werte gemeint) der Pixel geschaut werden. Das PNG-Format erlaubt es Bildern vier Farbkanäle zu haben [Roelofs, 1999]. Dabei steht je ein Band für den Rot-, Grün-, Blau- und Transparenz-Anteil. Erst wenn alle diese vier Kanäle "aufeinander gelegt" werden, entsteht das vollständige Bild [Boutell, 1997]. Entweder müssen also alle Farbkanäle nach Polygonen durchsucht und die gefundenen Polygone dann zusammengeführt werden, oder das Bild muss zu einem Band zusammengeführt werden. Letzteres bedingt aber eine Farbpalette, damit die Farbinformationen nicht verloren gehen.

Ein möglicher Ansatz, der von freien Open-Source-Geographisches-Informationssystemen wie Gdal[20], Osgeo[21], Qgis[22] und vielen anderen benutzt wird, ist es die benachbarten Pixel desselben Farbwertes als eine Region zu sehen und den Pixeln dieser Region dieselbe ID zu geben. Dazu geht der Algorithmus jeden Pixel durch und vergleicht ihn mit seinen vier (oben, unten, rechts und links) oder allen acht Nachbarn. Wenn eins der bisher betrachteten Nachbarn denselben Farbwert hat, bekommt der aktuell betrachtete Pixel die Region-ID dieses Nachbars. Wenn kein bisher betrachteter Nachbar denselben Farbwert hat, so wird ein neuer Polygon bzw. eine neue Region erstellt und dem aktuell betrachtetem Pixel wird eine neue Region-ID zugewiesen. Vor dem Algorithmus gibt es keine Region-IDs und am Ende des Algorithmus' gibt es soviele Region-IDs wie Polygone. Zur Übersicht wird der Algorithmus in Pseudocode beschrieben:

```
1  polygonize(GRAFIK)
2  Schritt 1: initialisiere regionCount bei 0
3
4  Schritt 2: Gehe alle Pixel durch
5
6  Schritt 2a: Weise dem ersten Pixel irgendeinen Wert fuer regionCount zu.
7
8  Schritt 2b: Wenn der naechste Pixel zu einem bereits durchlaufenem Pixel
       den selben Farbwert hat: uebernehme den regionCount
9       sonst: erhoehe regionCount um eins und weise diesen Wert dem neuen
            Pixel zu.
10
11 Schritt 3: Schreibe die Polygone in einen textbasierten Format
```

Nachdem alle Polygone identifiziert wurden, können sie serialisiert werden und sind

[20] http://gdal.org - Zugriff: 14.01.2015
[21] http://www.osgeo.org/ - Zugriff: 14.01.2015
[22] http://qgis.org - Zugriff: 14.01.2015

somit in einer textbasierten Datei, also liegen sie in Vektordaten vor (siehe Kapitel 3.2). Dieses Verfahren sorgt für sehr genaue Polygone, da jedes Pixel betrachtet wird

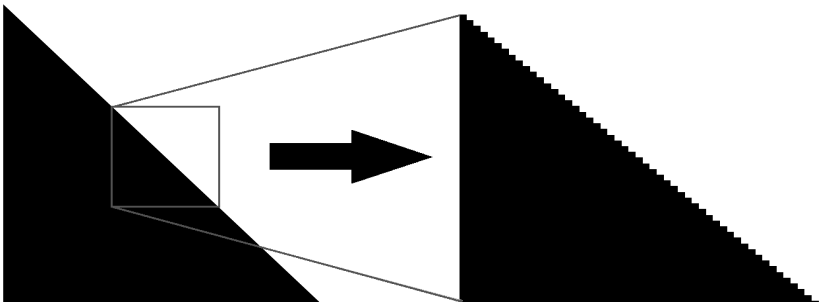

**Abbildung 7:** *Verdeutlichung des Treppeneffekts [Quelle: Eigene Darstellung]*

und nicht übersprungen oder vereinfacht wird. Das ist aber gleichzeitig ein Nachteil, da so der "Treppeneffekt" in die Vektordatei übernommen wird. Und da jede Stufe einen Eckpunkt des Polygons darstellt, wird aus einem dreieckigem Polygon, welches drei Eckpunkte haben müsste, ein n-eckiges Polygon, wobei die Zahl n von den Stufen abhängt, wie man in Abbildung 7 sehen kann. Die Ursache dafür liegt in der Struktur von Rastergrafiken. Sie besteht aus einer rasterförmigen Anordnung von quadratischen Pixeln. Um den Treppeneffekt, der in die Vektordatei übernommen wird, zu beseitigen, werden in Kapitel 4.2.2 zwei Kurvenglättungsalgorithmen vorgestellt. Es ist wichtig, dass dieses Problem gelöst wird, da die überschüssigen Punkte die Vektordaten erheblich aufblähen und somit die Effizienz der Weiterverarbeitung schwächen.

## 4.2 Korrektur

Im letzten Unterkapitel (4.1 Extraktion) wurden die Rasterdaten in Vektordaten umgewandelt. Die Umwandlungen haben jedoch unerwünschte Nebeneffekte zur Folge, wie z.B. den in Abbildung 7 dargestellten Treppeneffekt. Um die Fehler, die durch die Umwandlung entstanden sind oder bereits in den Ursprungsdaten lagen und von dort übernommen wurden, zu beseitigen, werden in den folgenden zwei Abschnitten Algorithmen vorgestellt.

### 4.2.1 Aussiebung der kleinen Polygone

Greift der WMS-Server auf verschiedene Datenquellen zu, um eine Anfrage zu erfüllen kann es sein, dass die Geodaten in verschiedenen Koordinatensystemen vorliegen. In so einem Fall müssen die Geodaten transformiert werden, damit sie im selben

Koordinatensystem vorliegen. Dies geschieht meistens automatisch und ohne, dass der Nutzer es mitbekommt. Bei der Transformation kann es aber zu kleinen Abweichungen kommen. So kann zwischen zwei angrenzenden Polygonen eine Lücke entstehen. Diese Lücke bildet ein Polygon und wird infolgedessen ebenfalls serialisiert. Eine andere Ursache für diese Lücken kann an der unterschiedlichen Art der Generierung der Geodaten liegen (siehe Kapitel 3). Da diese Abweichungen und somit die daraus resultierenden Polygone im Vergleich zu den eigentlichen Polygonen sehr klein sind und bei der Visualisierung in kleineren Zoomstufen kaum erkennbar sind, können sie entfernt werden, ohne dass es einen erkennbaren visuellen Unterschied gibt. Die Aussiebung der kleinen Polygone verkleinert die Vektordaten und die folgenden Operationen auf diesen Vektordaten werden schneller.
Um kleine Polygone entfernen zu können, müssen die Flächeninhalte aller Polygone berechnet werden und jene, die eine kleinere Fläche als einen bestimmten Wert haben, gelöscht werden. Eine weit verbreitete Methode hierfür ist es die Polygone in Dreiecke zu zerteilen, um dann ihre Fläche mittels trigonometrischen Funktionen zu ermitteln [Braden, 1986]. Die Summe der Flächen der Dreiecke eines Polygons entspricht seinem Flächeninhalt. Eine einfachere Methode, um den Flächeninhalt eines Polygons zu berechnen, ist die Gaußsche Trapezformel und geht zurück auf Carl Friedrich Gauss, der sie 1795 beschrieb [Braden, 1986]. Der Algorithmus benötigt

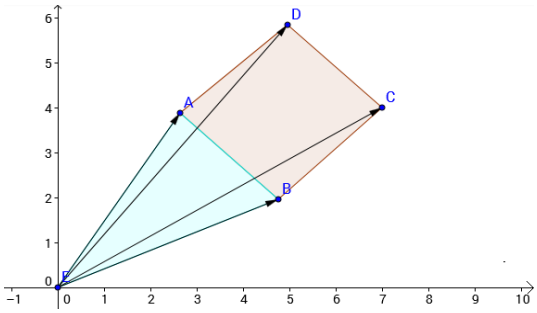


**Abbildung 8:** *Der Flächeninhalt des Vierecks $ABCD$ ergibt sich aus den Flächeninhalten der Dreiecke $\triangle BCE, \triangle CDE, \triangle DAE$ minus dem Dreieck $\triangle ABE$ [Quelle: Eigene Darstellung]*

die Koordinaten $(x_1, y_1), ..., (x_n, y_n)$ eines Polygons mit $n \in \mathbb{N}$ Eckpunkten auf einer Ebene. Dabei müssen die Koordinaten geordnet sein, oder mit anderen Worten im oder gegen den Uhrzeigersinn aufgezählt sein [Kedlaya, 2006].
Der Grundgedanke ist folgender: Man rechnet die Flächeninhalte der Dreiecke, die durch die Endpunkte jeder Kante und dem Koordinatenursprung entstehen, und subtrahiert dann die Dreiecke, die zwischen dem Polygon und dem Koordinatenursprung entstanden sind. Abbildung 8 verdeutlicht diesen Ansatz. Der Flächeninhalt des

Vierecks $ABCD$ ergibt sich aus den Flächeninhalten der Dreiecke $\triangle BCE, \triangle CDE, \triangle DAE$ minus das Dreieck $\triangle ABE$ Die Formel für dieses Verfahren lautet:
$A = \frac{1}{2} \mid \sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n \mid$
wobei $A$ die Fläche des Polygons, $n$ die Anzahl der Kanten des Polygons, und $(x_i, y_i)$ mit $i = 1, 2, ...n$ die Koordinaten der Eckpunkte des Polygons sind. Durch Umformungen erhält man [Rayner and Schmidt, 1969] [Rice and Knight, 1973] [Pretzsch, 2009]:

$$
\begin{aligned}
A &= \frac{1}{2} \mid \sum_{i=1}^{n} x_i (y_{i+1} - y_{i-1}) \mid = \frac{1}{2} \mid \sum_{i=1}^{n} y_i (x_{i+1} - x_{i-1}) \mid && (1) \\
&:= \frac{1}{2} \left| \sum_{i=1}^{n} x_i y_{i+1} - x_{i+1} y_i \right| = \frac{1}{2} \mid \sum_{i=1}^{n} det \begin{vmatrix} x_i & x_{i+1} \\ y_i & y_{i+1} \end{vmatrix} \mid && (2) \\
& \quad \text{wobei } x_{n+1} = x_1 \text{ und } x_0 = x_n \text{ (analog für } y) && (3)
\end{aligned}
$$

Der Faktor $\frac{1}{2}$ sorgt dafür, dass jeweils die Flächeninhalte der Dreiecke und nicht die der Parallelogramme berechnet werden, die entstehen, wenn man das Dreieck von dem Koordinatenursprung aus betrachtet nach hinten spiegelt. Der Betrag wird verwendet, um die Richtung der Koordinatensequenz (also im oder gegen den Uhrzeigersinn) zu ignorieren. Werden die Betragsstriche weggelassen, so müssen die Koordinaten gegen den Uhrzeigersinn angegeben sein [Kedlaya, 2006].

### 4.2.2 Polygon-Vereinfachung

Durch das in Kapitel 4.2 präsentierte Problem des "Treppeneffekts" werden die Vektordaten mit unnötigen Daten beschrieben. Beispielsweise hat ein Dreieck in einer Vektordatei drei Punkte, die die Eckpunkte darstellen (vier, wenn der erste und der letzte Punkt eines Polygons laut Dateiformat der selbe sein müssen). Durch den Treppeneffekt können daraus aber sehr viele Punkte werden, was von der Auflösung der Rasterdatei abhängt.
In diesem Unterabschnitt wird zuerst der Douglas-Peucker-Algorithmus [Douglas and Peucker, 1973] und anschließend ein radiale Entfernungen benutzender Algorithmus vorgestellt.

**Douglas-Peucker-Algorithmus** Der Douglas-Peucker-Algorithmus wurde im Jahre 1973 von David Douglas und Thomas Peucker vorgestellt und ist ein Algorithmus, der ein Polygon und eine spezifizierte Toleranz benötigt und das gegebene Polygon

mit der gegebenen Toleranz vereinfacht [Aitchison, 2012].
Gegeben ist eine Menge $V$, die alle Punkte eines Polygons enthält, und eine Toleranz $t$. Der Algorithmus gibt eine Menge $V'$ zurück, wobei $V' \subseteq V$. $V'$ enthält alle Punkte des vereinfachten Polygons, was im Extremfall zu einem Dreieck vereinfacht wurde. Der Algorithmus wurde entwickelt, um offene Polygonzüge (Linien, die an ihren Endpunkten miteinander verbunden sind, aber keine geschlossene Fläche bilden) zu vereinfachen.
Ein Polygon ist ein geschlossener Polygonzug und somit liegen der erste und der letzte Punkt aufeinander. Zunächst werden die Endpunkte miteinander verbunden und in die Menge $V'$ aufgenommen. Im Falle eines Polygons wird der erste und der vorletzte Punkt genommen, um einen offenen Polygonzug zu erhalten. Diese zwei Punkte werden verbunden, um eine Strecke zu erhalten, die **Ankerstrecke** genannt wird [van Kreveld et al., 1997]. Anschließend wird der Eckpunkt $v \in V$ bestimmt, der den größten Abstand $a$ zu der **Ankerstrecke** hat. Der Abstand wird mit der Toleranz $t$ verglichen und falls $t \geq a$ gilt, terminiert der Algorithmus mit dem ersten und dem letzten Punkt des offenen Polygonzugs, da kein Punkt weit weg genug ist, um die Toleranz zu überwinden. Im Falle eines Polygons wird der Punkt, der den größten Abstand zur *Ankerlinie* hat, der Ergebnismenge $V'$ hinzugefügt, um ein Polygon (Dreieck) zu erhalten. Gilt hingegen $t < a$, so wird $v$ in $V'$ aufgenommen und jeweils eine Strecke vom ersten Punkt zu $v$ und von $v$ zum (vor)letzten Punkt gebildet, sodass jetzt zwei Strecken vorhanden sind und die bisherige *Ankerstrecke* verschwindet. Dieses Verfahren wird rekursiv auf die neu gebildeten Strecken angewandt, bis alle Punkte $v' \in V$, deren $a > t$ ist, in $V'$ aufgenommen werden.
Zur Übersichtlichkeit und Veranschaulichung nochmal auf Polygone zugeschnitten in Pseudo-Code [Mokrzycki and Samko, 2012] und Abbildung 9:
Gegeben ist ein Polygon $P_1$ mit den Punkten $n_1, n_2, ...n_p$, wobei $n_1 = n_p$. Gesucht ist ein Polygon $P_2$ mit Punkten von $P_1$, deren Abstand zur Ankerstrecke die Toleranz $t$ überschreiten.

```
1 simplify({p_1,...,p_n},t)
2 Schritt 0: initialisiere Menge V* bei {} und setze Menge V = {v_1
      ,...,v_(n-1)} (Abbildung 5;0)
3
4 Schritt 1a: Verbinde v_1 und v_(n-1) zu Strecke a und fuege sie
      der Menge V* hinzu.
5 Schritt 1b: Ermittle v_c aus V mit dem maximalen Abstand zur
      Strecke a und fuege v_c in V* ein.
```

[23] http://upload.wikimedia.org/wikipedia/commons/thumb/9/91/Douglas_Peucker.png/220px-Douglas_Peucker.png (letzter Zugriff: 08.02.2015

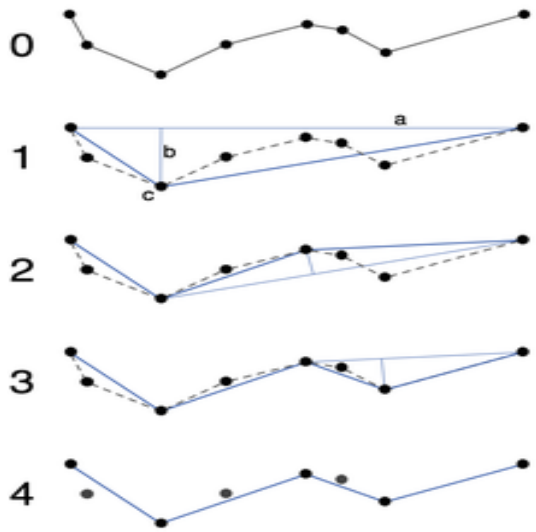


**Abbildung 9:** *Linienglättung nach Douglas-Peucker-Algorithmus [Quelle: [23]]*

```
6
7  Schritt 2: Bilde Strecken zwischen den benachbarten Punkten aus V
       *. (Abbildung 5;1)
8
9  Schritt 3a: Setze nacheinander alle Strecken als Ankerstrecke
10 Schritt 3b: Ermittle v_c aus V mit dem maximalen Abstand zur
       Strecke a und vergleiche Abstand mit Toleranz t
11 Schritt 3c: Wenn t >= Abstand: fahre mit der naechsten Strecke
       fort. Sonst: Fuege v_c in V* und mache rekursiv weiter
```

**Radiale-Entfernung-Algorithmus** Der Radiale-Entfernung-Algorithmus (*engl.: radial distance algorithm*) ist ein auf Entfernungen basierender Algorithmus. Ähnlich wie der Douglas-Peucker-Algorithmus erhält er ein Polygon und eine Toleranz und gibt einen Polygon aus, der vereinfacht wurde, oder der unverändert blieb, weil die Toleranz zu groß war. Anders als der Douglas-Peucker-Algorithmus wird hier nicht ein globaler Top-Down-Ansatz gewählt, sondern ein iterativer Ansatz. Das heißt, dass in diesem Algorithmus mit einem Schlüsselpunkt angefangen wird und dann der Abstand zum nächsten Punkt (dem sog. Testpunkt) gemessen wird. Ist der Abstand unterhalb der Toleranz, wird der Testpunkt entfernt und der nächste Punkt wird zum Testpunkt ernannt. Ist der Abstand zum neuen Testpunkt oberhalb der Toleranz, wird der Testpunkt zum Schlüsselpunkt und der folgende Punkt wird zum

Testpunkt. Der Pseudecode dazu sieht wie folgt aus:[24]

```
1   function radialDistance(PointList[], Tolerance)
2     keyID= 0
3     while (keyID < PointList.length-1)
4       testID= keyID+1
5       while ( keyID < PointList.length-1 && distance( PointList[
            keyID ], PointList[ testID ] ) < Tolerance )
6         PointList.remove( testID )
7     keyID++
8   end
```

In *PointList[]* sind die Koordinaten der Eckpunkte des Polygons enthalten.

[24] http://web.cs.sunyit.edu/~poissad/projects/Curve/about_algorithms/radial.php - Zugriff: 25.03.2015

# 5 Implementierung und Evaluation

Der Weg von Web Map Service-Kartenbildern (WMS-Kartenbilder) zu textuellen Geodaten, mit denen das Gleiche darstellbar ist, besteht aus mehreren Schritten. Er beinhaltet mehrere Schritte aus verschiedenen Disziplinen der Informatik, wie z.B. die Geoinformatik oder die Bildbearbeitung. Der Prozess der Umwandlung von WMS-Kartenbildern zu textuellen Geodaten kann in folgende Schritte zusammengefasst werden:

1) Herunterladen und Bereinigung der WMS-Kartenbilder

2) Zusammenführung der Kartenbilder zu einer großen Karte

3) Serialisierung der Polygone auf der Karte zu textuellen Geodaten

4) Bereinigung und Korrektur der textuellen Geodaten

5) Visualisierung der textuellen Geodaten

Da wie bereits erwähnt verschiedene Disziplinen aus der Informatik zum Einsatz kommen, wurde auf verschiedene Bibliotheken zurückgegriffen. Konnten diese nicht benutzt werden, wurden die Bibliotheken entweder ergänzt oder eine eigene Implementierung benutzt.
In den kommenden Unterkapiteln werden die Ausgangsdaten beschrieben. Danach folgt eine Beschreibung, wie die einzelnen Schritte realisiert wurden, und im Anschluss daran werden Metriken vorgestellt, anhand derer die Auswahl des Algorithmus' zur Kantenglättung getroffen wird.

## 5.1 Ausgangsdaten

Bei der Entwicklung dieses Umwandlungsdienstes wurden die WMS-Server der Berliner Senatsverwaltung verwendet und folglich ist der Dienst überwiegend auf diese oder diesen ähnliche Daten zugeschnitten. Da der Web Map Service (WMS, siehe Kapitel 2.2) ein sehr verbreiteter Standard ist, wird der im Rahmen dieser Bachelorarbeit entwickelte Umwandlungsdienst auch für viele andere WMS-Server funktionieren, deren Daten vom Aufbau her ähnlich sind.
Die Daten stammen aus dem FIS-Broker der Senatsverwaltung für Stadtentwicklung[25]. Der FIS-Broker ist ein Geodatenkatalog und

[25] http://www.stadtentwicklung.berlin.de/geoinformation/fis-broker/ - Zugriff: 10.02.2015

> *„ein umfangreicher Geodatenkatalog bietet Karten, Pläne und andere Daten mit Raumbezug aus Berlin und Brandenburg. Hier kann räumlich, z.B. über Adressen und inhaltlich (z.B. nach Schlagworten) gesucht werden.".*

Der FIS-Broker hat verschiedene Datensätze auf WMS[26]- und WFS-Servern[27].
Die WMS-Datensätze des FIS-Brokers weisen einen hohen Grad an Homogenität in ihrer Struktur auf. Dies ermöglicht, dass einige Einstellungen des Umwandlungsdienstes fest vorgegeben sind und einige andere vom Nutzer gewählt werden müssen. Klickt man aus der Liste der WMS-Datensätze einen an, so werden ein Musterbild, allgemeine Informationen, Informationen über den Anbieter und technische Details angezeigt. Ganz unten ist ein Link, der zum aktuell ausgewählten WMS-Server die "GetCapabilities"-Funktion (siehe Kapitel 2.2) aufruft. Es werden stets die Versionen 1.0.0, 1.1.0, 1.1.1 und 1.3.0 angeboten, weshalb der Umwandlungsdienst hier keine Auswahl bietet, sondern standardmäßig Version 1.3.0 auswählt. Eine weitere Gemeinsamkeit sind die angebotenen Grafikformate. Zur Auswahl stehen JPG- und PNG-Grafiken, wobei der FIS-Broker selbst standardmäßig PNG-Grafiken anbietet. Da PNG-Grafiken eine Kompressionstechnik benutzen, die keinen Datenverlust gewährleistet [Furht, 2006b], und die Kompressionstechnik der JPG-Grafik aber für einen Datenverlust sorgt [Furht, 2006a], wählt der Umwandlungsdienst stets das PNG-Format. Ein Rauschen im Bild hätte nämlich zur Folge, dass während der Polygonisierung beim Übergang zwischen verschiedenen Polygonen fälschlicherweise zusätzliche Polygone erkannt werden. Es gibt aber auch einige Unstimmigkeiten zwischen der Informationsseite der WMS-Datensätze und der Informationsdatei, die die "GetCapabilities"-Funktion ausgibt. Die Informationsdatei zeigt an, dass neben dem Koordinatensystem EPSG:3068 (siehe Kapitel 3.2) das Koordinatensystem EPSG:4326 angezeigt wird. Wenn dem Nutzer die Wahl zwischen diesen beiden ermöglicht ist, ist die Auswahl des EPSG:4326 wahrscheinlicher, da es das Gängigere von beiden ist. Wird jedoch EPSG:4326 ausgewählt, wird intern immer noch mit EPSG:3068 gearbeitet. Das erkennt man daran, dass Kacheln, die von den EPSG:4326-Koordinaten her exakt benachbart sein müssten, leicht verschoben sind. Das sorgt für stark verzerrte und ungenaue Ergebnisse, die sich auf die weiteren Schritte der Polygonisierung auswirken. Aus diesem Grund wird die Wahl des Koordinatensystems aufgehoben und EPSG:3068 zum Standard des Umwandlungsdienstes erklärt.

[26] http://fbinter.stadt-berlin.de/fb/berlin/service.jsp?type=WMS - Zugriff: 10.02.2015
[27] http://fbinter.stadt-berlin.de/fb/berlin/service.jsp?type=WFS - Zugriff: 10.02.2015

## 5.2 Implementierung

Die fünf Schritte zur Realisierung des Umwandlungsdienstes können in die beiden Gruppen **“Frontend”** und **“Backend”** aufgeteilt werden. Im **“Frontend”** findet zu Beginn einer Umwandlungsoperation die Kommunikation mit dem Nutzer statt, der die Parameter für die Umwandlung bestimmt. Es besteht aus einer *HTML-Seite*, sowie *(Java-)Servlets*. Außerdem dient das **“Frontend”** zur Visualisierung von Geodaten, die in einer GeoJSON-Datei enthalten sind. Im **“Backend”** findet

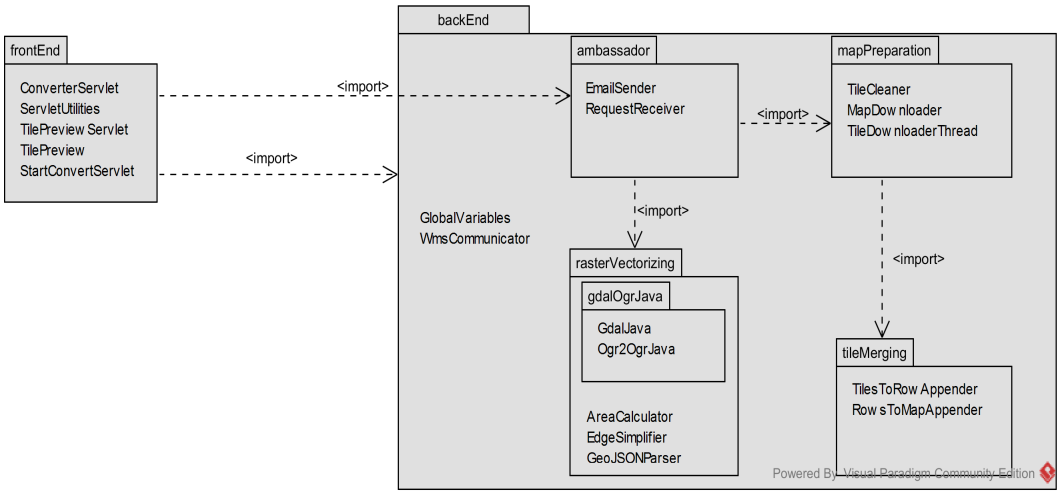


**Abbildung 10:** *Paket-Übersicht des Umwandlungsdienstes in UML-Notation*

die Umwandlung und die Benachrichtigung des Klienten via Email statt. Es ist aufgeteilt in mehrere *Java-Pakete*, die Klassen zur Realisierung der einzelnen Schritte der Umwandlung enthalten. Abbildung 10 zeigt noch einmal die Struktur des Umwandlungsdienstes.

### 5.2.1 Frontend

Das *Frontend* des Umwandlungsdienstes kann die zu GeoJSON-Dateien umgewandelten Geodaten auf einer web-basierten Karte anzeigen. Außerdem dient es als Schnittstelle zwischen dem Klienten und dem Server, sodass Umwandlungsanfragen über das Frontend gestartet werden.
Im Rahmen dieser Bachelorarbeit wurde das in den Abbildungen 11 und 12 dargestellte *Frontend* als eine *HTML*-basierte Web-Oberfläche realisiert, welche durch einen Tomcat-Webserver[28] (Version 8.0.20) bereitgestellt wurde. Tomcat ermöglicht die Verwendung von Webseiten mit dynamischem Inhalt mittels *Servlets*. *Servlets*

[28] http://tomcat.apache.org/ - Zugriff: 18.03.2015

sind Java-Klassen, deren Instanzen innerhalb eines Webservers Anfragen von Klienten entgegennehmen und beantworten können. Dabei kann die Antwort *HTML*-Code enthalten, sodass ein Austausch von Botschaften und Parametern zwischen den Klienten und dem Server stattfinden kann. Dies ließe sich auch mit *Java Server Pages* realisieren, jedoch eignen sich Servlets für Anwendungen mit einer hohen Laufzeit mehr [Basham et al., 2008]. Das Frontend besteht hauptsächlich aus dem "Catalogue"- und dem "Converter"-Teil.

**Catalogue** Das "Catalogue"-Menü ist gleichzeitig die Hauptseite und, wie in Abbildung 11 zu sehen ist, besteht es aus den Teilen:

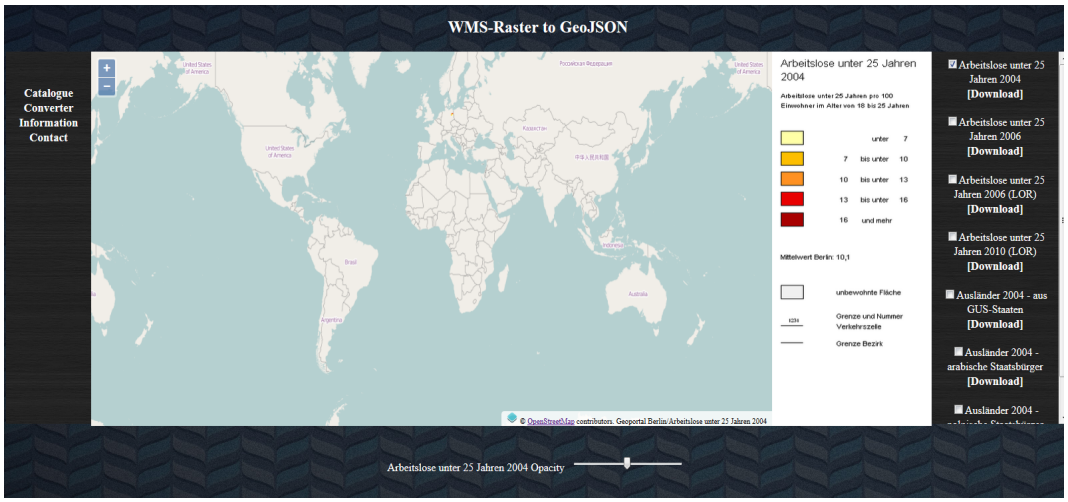


**Abbildung 11:** *Screenshot vom Catalogue-Teil der Web-Oberfläche des Frontends*

- Menüleiste (auf der linken Seite)
- Eine webbasierte Karte namens "Openlayers 3" (mittig)
- Eine Kartenlegende, die nur gezeigt wird, wenn eine Karte ausgewählt wird (erscheint auf der linken Seite der Karte)
- Ein Katalog der zur Verfügung stehenden Ebenen (auf der rechten Seite)
- Ein Regler, mit dem die Transparenz der zusätzlich gewählten Ebenen festgelegt werden kann (erscheint unter der Karte)

**Converter** Abbildung 12 zeigt den Converter-Teil der Seite. Ein Klick auf den "Converter"-Schriftzug in der linken Leiste führt zu diesem Teil. Zunächst ist nur

ein Textfeld zu sehen, in das die URL des WMS-Servers eingetragen werden muss. Wird anschließend auf den “analyse”-Knopf gedrückt, so wird erstmalig mit einem Servlet kommuniziert und ist der Link eine URL zu einem gültigen WMS-Server, so werden die zur Verfügung stehenden Optionen angeboten. Sind die gewünschten Optionen gewählt und die Emailadresse für die Erhaltung des Ergebnisprodukts eingetragen, kann auf den “preview”-Knopf gedrückt werden.

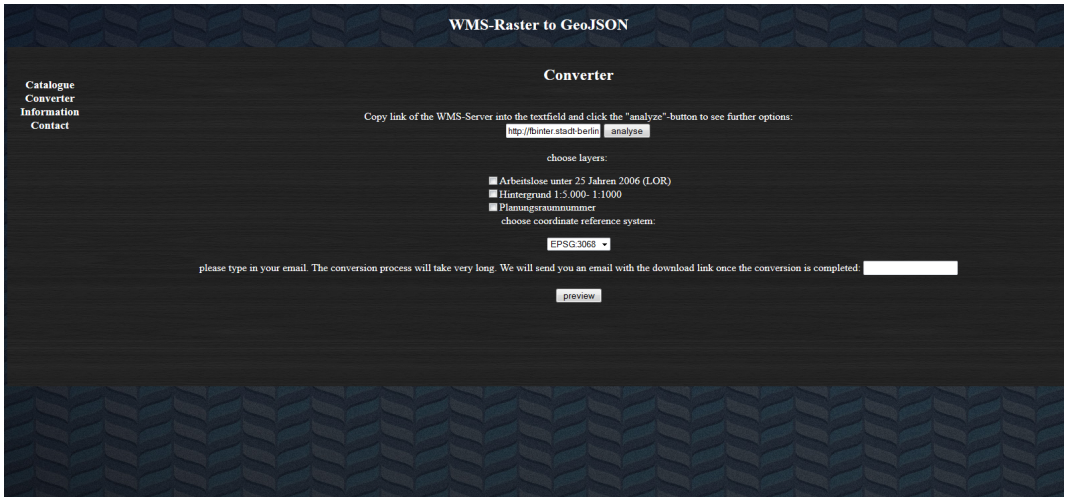


**Abbildung 12:** *Screenshot vom Converter-Teil der Web-Oberfläche des Frontends*

Nach einer erneuten Kommunikation mit einem Servlet wird hier ein Link zu einer Beispielkachel mit den ausgewählten Optionen angezeigt. Ist die Beispielkachel zufriedenstellend, kann auf den “convert”-Knopf gedrückt werden, um den Umwandlungsdienst zu starten. Andernfalls können entweder mit der “Zurück”-Funktion des Browsers oder dem “Converter”-Schriftzug in der linken Leiste Änderungen an den Optionen vorgenommen werden.

**Openlayers 3** Openlayers 3[29] ist eine open-source, auf Javascript basierende Bibliothek, die die Visualisierung von Karten ermöglicht. Dank verschiedener Klassen und Schnittstellen kann es aber auch benutzt werden, um Webanwendungen zu erzeugen, in denen eine Interaktion mit den angezeigten Karten möglich ist. Hinzu kommt, dass es viele Standards, wie z.B. das Web Map Service-Standard (siehe Kapitel 2.2), benutzt und dadurch eine größere Beliebtheit errungen hat [Burdziej, 2012].
In dieser Arbeit wurde Openlayers v3.3.0 benutzt, um die Karte sowie die umgewandelten Geodaten anzeigen zu können. Das *ol.Map*-Element bildet den Kern von

[29] http://openlayers.org/ - Zugriff: 18.03.2015

Openlayers 3. Ihm muss mindestens ein *ol.Layer*-Element, also eine Ebene zugewiesen werden. Die Ebene muss ein *ol.Source*-Element besitzen, damit es angezeigt werden kann. In dem *ol.Source*-Element wird die Quelle der Ebene festgelegt. Die Quelle kann sowohl eine Online-, als auch eine Offline-Quelle sein. In Abbildung 11 ist z.B. zu sehen, dass eine Open Street Map[30]-Ebene als Grundebene benutzt wird und die Geodaten aus dem Katalog als GeoJSON-Ebene raufgelegt werden.
Damit die Farben der Polygone der in GeoJSON umgewandelten Geodaten auch in Openlayers sichtbar sind, wurde eine Javascript-Funktion geschrieben, die den *Style*-Attribut der jeweiligen Ebene so ändert, dass die Farbe jedes Polygons aus der GeoJSON-Datei ausgelesen wird. Eine andere Funktion sorgt für das Erscheinen eines Reglers, sobald eine Ebene aus dem Katalog gewählt wird. Dieser Regler verändert den Grad der Transparenz der ausgewählten Ebene.

### 5.2.2 Backend

Im *backEnd*-Paket befinden sich die Klassen *WMSCommunicator*, sowie *GlobalVariables*. Es enthält ebenfalls die vier Pakete *ambassador*, *mapPreparation*, *tileMerging* und *rasterVectorizing*. Mit Ausnahme der Klasse *WMSCommunicator* werden alle anderen Klassen und Pakete erst benutzt, wenn die Umwandlung stattfindet. Die Aufgabe des *WMSCommunicator* ist es die Verbindung zu einem WMS-Server aufzubauen, dessen Link es erhalten hat. Mithilfe dieser Klasse können andere Klassen einige wichtige Werte, wie z.B. die angebotenen Ebenen oder Koordinatensysteme, herausfinden. *GlobalVariables* beinhaltet Verzeichnispfade zu externen Programmen und Bibliotheken.

**ambassador-Paket** Wie in der Abbildung 13 dargestellt wird dieses Paket vom *frontEnd*-Paket importiert und importiert seinerseits die Pakete *mapPreparation* und *rasterVectorizing* im *backEnd*-Paket. Das liegt daran, dass in dem *ambassador*-Paket sowohl die Kommunikation mit dem **"Frontend"**, als auch die Koordination der Umwandlungsschritte geschieht. Die *RequestReceiver*-Klasse wird direkt vom *StartConvertServlet* aus mit den nötigen Parametern für die Umwandlung erzeugt. Der *RequestReceiver* greift auf das *mapPreparation*-Paket zu, um die gesamte Karte des WMS-Servers als eine Bilddatei zu haben. Diese Bilddatei wird anschließend dem Paket *rasterVectorizing* übergeben, der die Polygone serialisiert, die Kanten glättet, kleine Polygone entfernt und eine Koordinatentransformation durchführt.

[30] http://www.openstreetmap.org - Zugriff: 18.03.2015

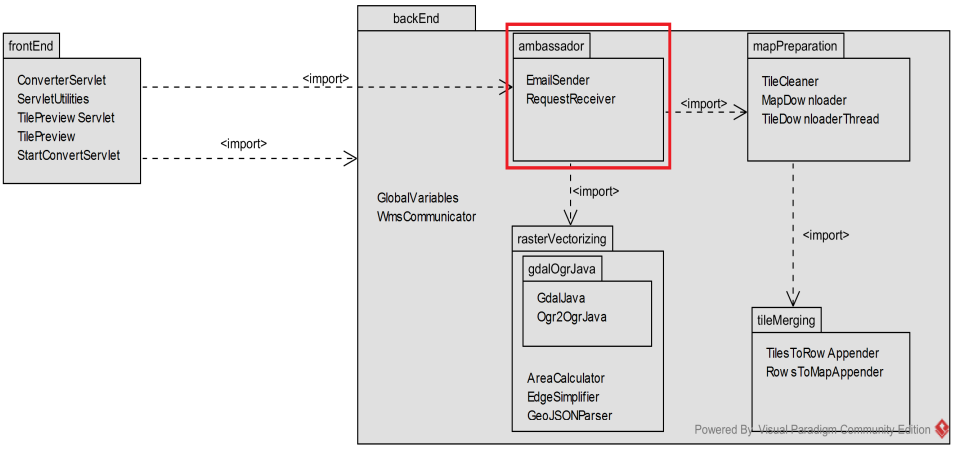


**Abbildung 13:** *Das ambassador-Paket in der Paketstruktur des Umwandlungsdienstes in UML*

Zum Schluss erhält das *RequestReceiver*-Paket die finale GeoJSON-Datei, welche dann mit Hilfe der *EmailSender*-Klasse an die Emailadresse des Klienten geschickt wird.

**mapPreparation-Paket** Das *mapPreperation*-Paket enthält Klassen, die die Kacheln des WMS-Server herunterladen (*TileDownloaderThread*), die die Kacheln von schwarzen Begrenzungen entfernen (*TileCleaner*) und als Schnittstelle zwischen den Paketen *ambassador* und *tileMerging* dienen (*TileDownloader*).
Die Abbildung 14 zeigt, dass es das *tileMerging*-Paket importiert und vom *ambassador*-Paket importiert wird. Es bekommt die erforderlichen Parameter, wie z.B. die URL des WMS-Servers und die Koordinaten. Mit diesen Informationen erstellt die *MapDownloader*-Klasse Threads (TileDownloaderThreads), die jeweils zehn horizontal benachbarte Kacheln herunterladen. Diese Kacheln werden jeweils von der Klasse *TileCleaner* gesäubert (siehe Kapitel 4.1.1) und anschließend dem *tileMerger*-Paket übergeben, damit es die benachbarten Kacheln zu einen von zehn Reihen zusammenfügt. Immer wenn so eine Reihe entsteht, wird sie dem Ergebnisbild hinzugefügt, sodass nach und nach die ganze Karte entsteht. Die Aufteilung der Karte in 100 Kacheln, die zu zehn Reihen und anschließend einer Karte zusammengefügt werden erfolgt, weil die WMS-Server der Berliner Senatsverwaltung[31] keine Anfragen bearbeiten, in denen große Kacheln bei einer hohen Auflösung gefordert werden.

[31] http://fbinter.stadt-berlin.de/fb/berlin/service.jsp?type=WMS - Zugriff: 17.03.2015

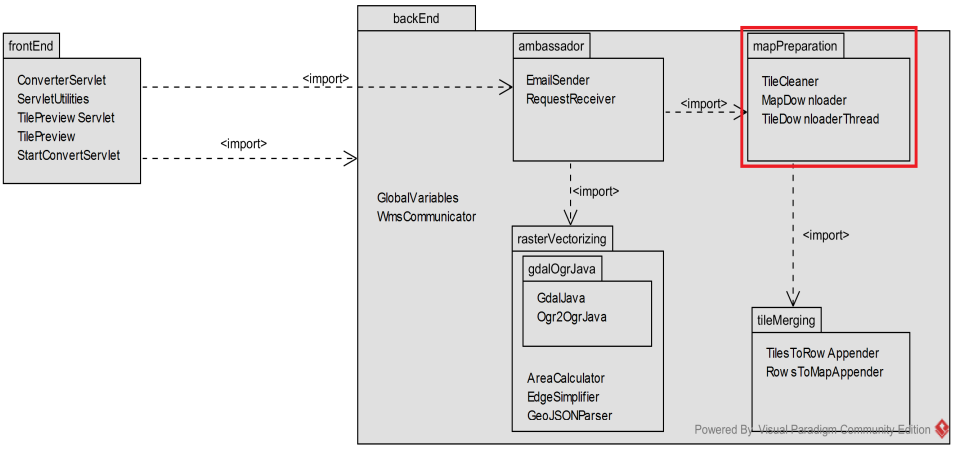


**Abbildung 14:** *Das mapPreparation-Paket in der Paketstruktur des Umwandlungsdienstes in UML*

Die *TileDownloaderThread*s laden die Kacheln gleichzeitig herunter, bevor sie nebenläufig gesäubert werden. Damit wird eine effizientere Nutzung der Computerressourcen und ein geringerer Zeitverbrauch erreicht. Dieses Paket bildet somit den ersten Schritt ( 1. Herunterladen und Bereinigung der WMS-Kartenbilder) des Umwandlungsprozesses. Die fertige Karte wird dann dem *ambassador*-Paket zur weiteren Verarbeitung übergeben.

**tileMerging-Paket** Das *tileMerging*-Paket hat die beiden Klassen *TilesToRowAppender* und *RowsToMapAppender*. Beide Klassen benutzen die *im4java*-Bibliothek[32], bei der es sich um eine Java-Schnittstelle zum Bildbearbeitungstool ImageMagick[33] handelt.

[32] http://sourceforge.net/projects/im4java/ - Zugriff: 17.03.2015
[33] http://www.imagemagick.org/ - Zugriff: 17.03.2015

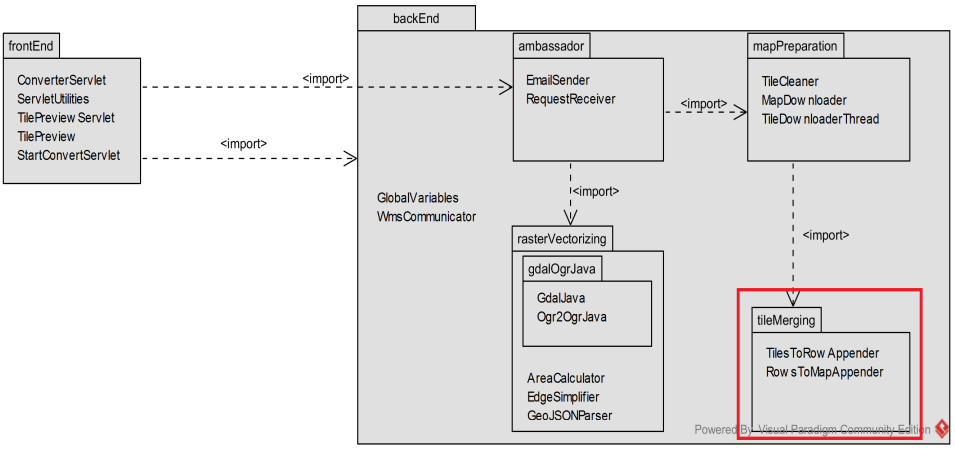


**Abbildung 15:** *Das tileMerging-Paket in der Paketstruktur des Umwandlungsdienstes in UML*

Der Abbildung 15 ist entnehmbar, dass dieses Paket keine weiteren Pakete mehr importiert. Es wird lediglich vom *mapDownloader*-paket benutzt, um die Kacheln zu Reihen und diese Reihen dann zu einer Karte zusammenzuführen. Da die zusammenzufügenden Bilder sehr groß sind (,beispielsweise die fertige Karte 49500 x 40900 Pixel groß sein) ist der Speicherverbrauch sehr groß. Es wird der Q8-build von ImageMagick 6.8.9 benutzt, der pro Pixel 4 Byte für die temporären Dateien benötigt. Wenn das Beispiel von vorhin erneut herangezogen wird, entsteht ein Speicherverbrauch von 49500 x 40900 x 4 $\approx$ 8,1 Gigabyte. Stehen dem Server nicht so viel Arbeitsspeicher und/oder Festplattenspeicher zur Verfügung, kann in der *GlobalVariables*-Klasse aus dem *backEnd*-Paket ein Pfad zu einem Speicherort mit genügend Kapazität festgelegt werden. Dieses Paket übernimmt den 2. Schritt des Umwandlungsdienstes (Zusammenführung der Kartenbilder zu einer großen Karte).

**rasterVectorizing-Paket** In diesem Paket befinden sich Klassen und ein weiteres Paket, die dazu da sind, die Polygone auf der Karte zu serialisieren und die dadurch entstandenen Daten zu georeferenzieren und zu bereinigen. Ein Blick auf die Abbildung 16 erklärt, dass auf das Paket vom *ambassador*-Paket aus zugegriffen wird. Dies geschieht durch die Klasse *RequestReceiver*, die die verschiedenen Klassen im *rasterVectorizing*-Paket aufruft. Um ihre Aufgaben zu erfüllen, rufen sich die Klassen dieses Pakets auch gegenseitig auf. Nachdem die Karte fertiggestellt wurde, wird das *gdalOgrJava*-Paket benutzt, um die Polygone der Karte zu serialisieren.

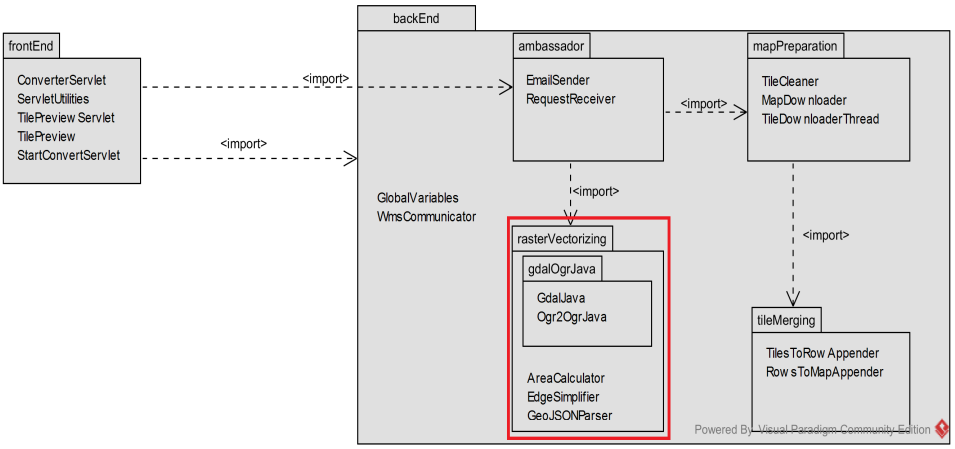


**Abbildung 16:** *Das rasterVectorizing-Paket in der Paketstruktur des Umwandlungsdienstes in UML*

Als Ergebnis entsteht eine GeoJSON-Datei (siehe Kapitel 3.2.2), die die Polygone in text-kodierter Form enthält, wobei die Koordinaten der Eckpunkte, durch die die Polygone definiert werden, sich auf das Bild beziehen. Dadurch ist der Ursprung des Koordinatensystems die linke obere Ecke der Karte. Um die ursprünglichen Geokoordinaten der Polygone zu ermitteln, wird diese GeoJSON-Datei der *GeoJSONParser*-Klasse übergeben. Ist das getan, kann die Kantenglättung vollzogen werden um das Problem, welches in Kapitel 4.2 beschrieben wird, zu beseitigen. Die Klasse *EdgeSimplifier* ist dieser Aufgabe gewidmet. Es macht Gebrauch von einer Java-Bibliothek namens simplify-java[34], in der der Algorithmus aus dem Kapitel 4.2.2 implementiert ist. Ist die Kantenglättung auch erledigt, kann die GeoJSON-Datei, in der sich sehr viele die Ergebnisdatei unnötig vergrößernde kleine Polygone befinden, (siehe Kapitel 4.2) von diesen kleinen Polygonen gesäubert werden. Diese Aufgabe übernimmt die *AreaCalculator*-Klasse.

**GdalOgrJava** Um die Karte zu "polygonisieren" wird eine Java-Bibliothek benutzt, die auf eine Sammlung von kleinen Programmen für Geoinformationssystemen namens "gdal/ogr"[35](Version 1.11.1) zugreift.

## 5.3 Evaluierungsmetriken

Einer der wichtigsten Aspekte dieses Umwandlungsdienstes ist, die Kantenglättung bzw. Polygon-Vereinfachung, denn dadurch wird der sogenannte Treppeneffekt (sie-

[34]https://github.com/hgoebl/simplify-java - Zugriff: 17.03.2015
[35]http://www.gdal.org/ - Zugriff: 17.03.2015

he Kapitel 4.2) neutralisiert. Das hat zum einen den Vorteil, dass die Daten korrekter sind, da nun diagonale Linien abgebildet werden können, wo vorher "Treppenstufen" waren. Zum anderen hat es den Vorteil, dass die Polygone viel weniger Eckpunkte haben und dadurch die Speicherbelegung der Vektordaten um bis zu 90% verringert wird. In Kapitel 4.2.2 wurden zwei Algorithmen vorgestellt, die die Polygone vereinfachen. In diesem Abschnitt findet eine Evaluation statt, um eine Auswahl zwischen den beiden Algorithmen zu fällen. Dabei werden einige Kriterien sowie ihre Metriken vorgestellt, anhand die Algorithmen verglichen werden.
Um einen besseren Vergleich ziehen zu können, wurde extra ein Java-Programm geschrieben, dem beim Aufruf eine WMS-Server-URL, sowie das zu verwendende Kantenglättungsalgorithmus übergeben wird. Im Gegensatz zum Umwandlungsdienst werden alle Kacheln nacheinander heruntergeladen und sie werden nicht zusammenfügt, sondern direkt nach Entfernung der schwarzen Pixel polygonisiert. Anschließend werden die Koordinaten in den erzeugten GeoJSON-Dateien in das ursprüngliche Koordinatensystem der Karte projiziert. Ist das getan, werden entweder mit dem Douglas-Peucker-Algorithmus, oder mit dem Radiale-Entfernung-Algorithmus (*radial distance algorithm*) die Polygone vereinfacht, bevor die kleinen Polygone ausgesiebt werden. Ist der Hauptteil der Polygonbearbeitung abgeschlossen, werden die Koordinaten zurück zu den ursprünglichen Bildkoordinaten projiziert, damit die Textdatei in der sich die Polygone befinden, in eine Bilddatei umgewandelt werden kann, die diese Polygone darstellt. Diese neu erstellten Bilder werden benutzt, um sie mit den originalen Kacheln zu vergleichen. Sämtliche Grafiken wurden mit plotly[36] erstellt.
Damit eine höhere Fairness zwischen den Algorithmen herrscht, wurden beide Algorithmen mit der selben Toleranz und den selben Daten getestet. Um eine höhere Aussagekraft zu erhalten, wurde das Evaluierungsprogramm mit drei verschiedenen Datensätzen gestartet:

**Datensatz 1: Arbeitslose unter 25 Jahren 2004**[37]

**Datensatz 2: Senioren 2008 (LOR)**[38]

**Datensatz 3: Einwohnerdichte 2010 (Umweltatlas)**[39]

[36] http://plot.ly - Zugriff: 29.03.2015

[37] http://fbinter.stadt-berlin.de/fb/berlin/service.jsp?id=MsozSarblos252004senstadt&type=WMS - Zugriff: 29.03.2015

[38] http://fbinter.stadt-berlin.de/fb/berlin/service.jsp?id=MsozS_E6_2008LORsenstadt&type=WMS - Zugriff: 29.03.2015

[39] http://fbinter.stadt-berlin.de/fb/berlin/service.jsp?id=k06_06ewdichte2010senstadt&type=WMS - Zugriff: 29.03.2015

**Vergleich der Bilder:** Damit die Textdateien, die im GeoJSON-Format vorliegen in Bilder umgewandelt weren können, wurde ein Programm geschrieben, welches Textdateien ins SVG-Format (*Scalable Vector Graphics*[40]) überführen. Das SVG-Format basiert auf der XML-Sprache und ist eine Spezifikation zur Beschreibung zweidimensionaler Vektorgrafiken [Dahlström et al., 2011]. Die SVG-Dateien werden mithilfe der Apache™ Batik SVG Bibliotheken[41] in das PNG-Format umgewandelt. Zunächst wurden zwei Kriterien ausgewählt, mit Hilfe derer die Originalbilder mit den veränderten Bildern verglichen werden sollten. Ein Kriterium ist der Anteil an Pixeln, die sich geändert haben. Der Gedanke bei diesem Kriterium war es am Ende des Evaluierungsverfahrens dem Algorithmus, dessen Ergebnisbilder einen höheren Anteil an Pixeln mit den Originalbildern gemeinsam haben zu favorisieren. Dieses Kriterium hat sich als zu schwach ergeben, da in beiden Fällen ein Unterschied von <0,0001% verzeichnet wurde. Das zweite Kriterium spielt bei der Evaluation durchaus eine Rolle. Es vergleicht nämlich die Farbwerte jedes Pixels beider Bilder. Dazu wird der *RGB-Wert* der Pixel genommen und die Rot-Grün-Blau-Kanäle verglichen. Je ähnlicher also die Zusammensetzungen der Farbwerte aus den drei Farbkanälen sind, desto kleiner sind die Unterschiede zwischen diesen Pixeln.

Die Abbildungen 17, 18, 19 zeigen die Unterschiede der Pixelwerte zwischen dem Original- und dem veränderten Bild. Auf der X-Achse sind die Nummern der Kacheln aufgelistet. Eigentlich sind es in allen drei Datensätzen 100 Kacheln gewesen, allerdings wurden jene Kacheln, die sich weniger als 5% geändert haben, der Übersichtlichkeit wegen rausgenommen.

[40] http://www.w3.org/Graphics/SVG/ - Zugriff: 29.03.2015
[41] https://xmlgraphics.apache.org/batik/ - Zugriff: 29.03.2015

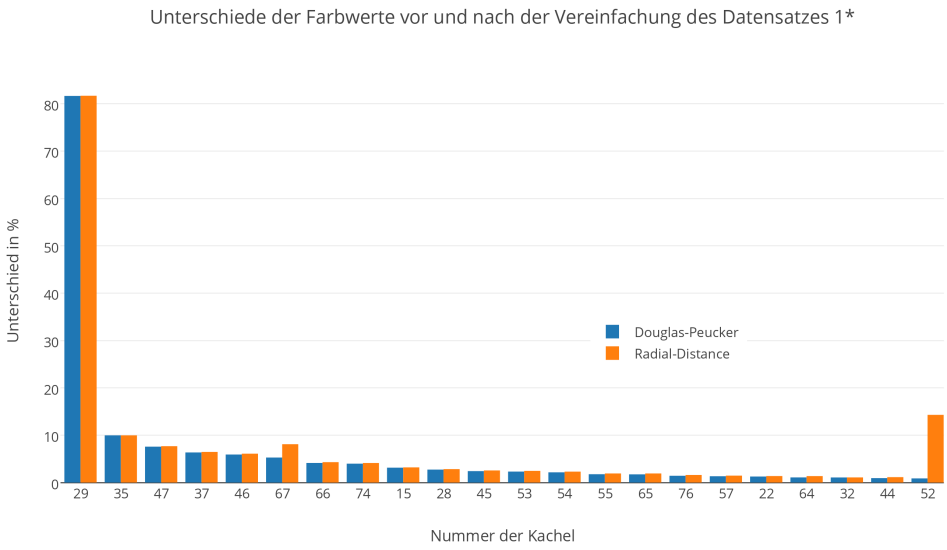


**Abbildung 17:** *Unterschiede der Farbwerte vor und nach der Vereinfachung des Datensatzes 1**

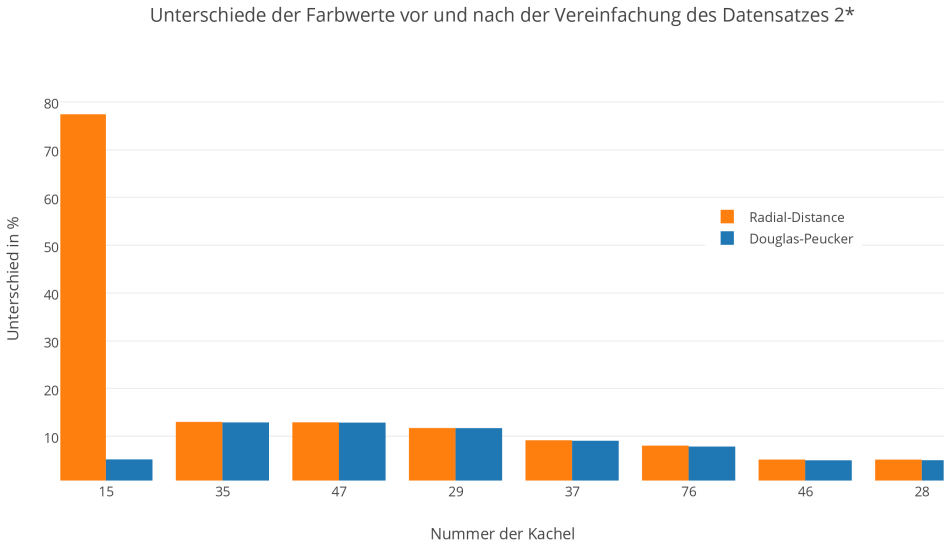


**Abbildung 18:** *Unterschiede der Farbwerte vor und nach der Vereinfachung des Datensatzes 2**

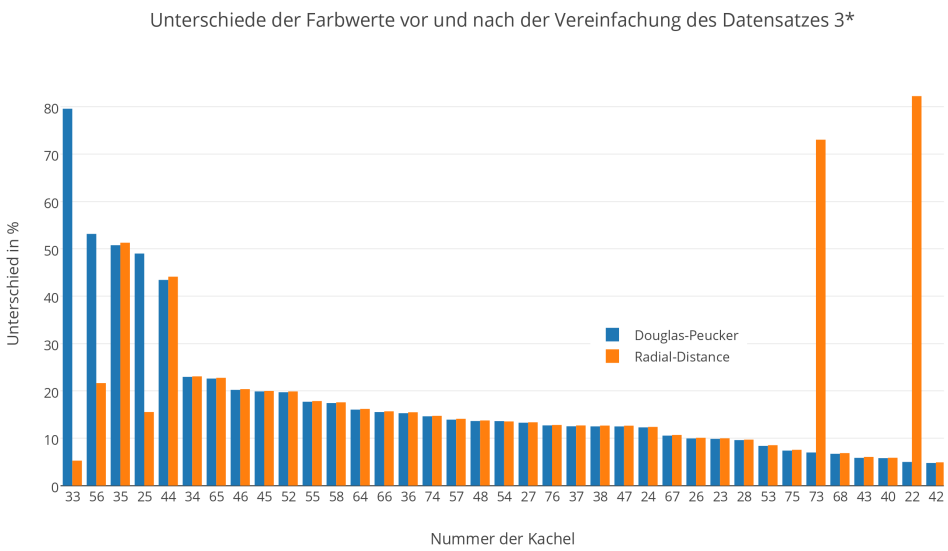


**Abbildung 19:** *Unterschiede der Farbwerte vor und nach der Vereinfachung des Datensatzes 3**

Die Y-Achse zeigt den Unterschied der RGB-Zusammensetzung der Pixel zwischen der Kachel vor und nach der Bearbeitung. Idealerweise sollten die Balken möglichst klein sein, da dies eine geringe Veränderung des Bildes bedeutet. Die Balken dürfen aber auch nur dann sehr klein sein bzw. nicht-existent sein, wenn die Polygone bereits einfach genug sind oder es ein Bild ist, in der es keine Polygone gibt (z.B. weil das Bild am Rand der Karte liegt). In der Regel wird der Unterschied zwischen den Bildern größer, je mehr und je komplexer die Polygone auf den Kacheln sind. Denn in diesen Fällen gibt es mehr zu vereinfachen. Den Abbildungen 17, 18, 19 sind drei auffallende Merkmale entnehmbar:

1) In allen drei Durchläufen hat der Douglas-Peucker-Algorithmus, wenn auch sehr geringfügig, besser abgeschnitten.

2) Die horizontal mittigen Kacheln wurden am meisten verändert.

3) In allen drei Diagrammen sind Ausreißer.

Ergänzung zu 2): Das ist keine Überraschung, da in diesen Kacheln aufgrund der höheren Bevölkerungsdichte die größte Konzentration an kleinen, dafür aber zahlreichen Polygonen ist. Und wie bereits erwähnt wurde, bedeutet das, dass es mehr potenziell zu verändernde Polygone gibt.

Ergänzung zu 3): Die Ausreißer sind auf einen Fehler während der Extraktion der Polygone zurückzuführen. Bei der Extraktion der Polygone ist die Serialisierung des weißen Polygons, der alle anderen weißen Polygone umfasst (sozusagen der "Hintergrund"), fehlgeschlagen. Dadurch hat dieses Polygon in der Vektordatei gefehlt. Als die Vektordatei im SVG-Format in ein PNG-Bild konvertiert wurde, wurden die Stellen, die in den Originalkacheln weiß dargestellt wurden, nun transparent. Daher rührt der große Unterschied zwischen diesem und dem ursprünglichen Bild, was sich natürlich in der Statistik stark bemerkbar macht.

**Ressourcenverbrauch:** Neben der Qualität der Ergebnisse ist der Ressourcenverbrauch ebenfalls ein wichtiger Aspekt bei der Wahl zwischen verschiedenen Algorithmen. Dieser Umwandlungsdienst ist dafür ausgelegt, als ein Server Anfragen entgegenzunehmen und diese zu bearbeiten. Aus dem Grund ist eine möglichst effiziente Nutzung der Ressourcen notwendig. Ein geringerer Ressourcenaufwand bedeutet bei effizienter Nutzung der Ressourcen eine schnellere Bearbeitung und dadurch geringere Kosten für den Betreiber. Außerdem sorgt eine schnellere Bearbeitung für eine höhere Zufriedenheit beim Klienten [Chandler and Hyatt, 2002]. Der Ressourcenverbrauch wurde aufgezeichnet, während das Programm zum Vergleich der Bilder lief. Dadurch wurde nicht nur der Ressourcenverbrauch des jeweiligen Algorithmus' aufgezeichnet, sondern auch die der Programme, die die Vektordaten nach der Polygon-Vereinfachung noch weiter bearbeiten. Das ist notwendig, da sich der Ressourcenverbrauch dieser Schritte, je nach Ergebnis des Kantenglättungsalgorithmus verändern kann. Die Eigenschaften des Systems auf dem der Ressourcenverbrauch Gemessen wurde, sind in Tabelle 3 aufgelistet.

| | |
|---|---|
| Prozessor | Intel(R) Core(TM) i7-2620M CPU @ 2.7GHz |
| Arbeitsspeicher | 8 GB (DDR 3) |
| phys. Speicher | $\sim$ 20 GB freier Speicher (SSD) |
| Betriebssystem | Windows 7 Home Premium (64 Bit) |

**Tabelle 3:** *Systemeigenschaften des Test-Rechners*

Die Daten wurden mit dem bereits in Windows vorinstalliertem Programm "Leistungsüberwachung" (perfmon.exe) gesammelt. Es wurden die Belegung des Arbeitsspeichers und die Nutzung des Prozessors gemessen. Dazu wurden in perfmon.exe sogenannte "benutzerdefinierte Sammlungssätze" erstellt. Ein Sammlungssatz zur CPU-Überwachung und ein Sammlungssatz zur Arbeitsspeicherüberwachung. Es wurde die Aktivität der Prozesse java.exe, compare.exe und convert.exe gemessen. Der Prozess java.exe führt das Java-Programm aus, compare.exe sowie convert.exe

sind Teil von ImageMagick und werden vom Java-Programm aufgerufen, um die Bilder zu vergleichen bzw. zu konvertieren. Die in perfmon.exe ausgewählten Parameter sind Prozess -> Prozessorzeit in %, Prozess -> Arbeitsseiten - privat und Prozess -> Auslagerungsseiten (byte). Die Leistungsüberwachung erstellt jeweils eine Logdatei für die CPU- und eine für die Arbeitsspeicheraktivität eines Prozesses. Es gibt drei Datensätze und zwei Algorithmen, sodass insgesamt 12 Logdateien entstanden sind. Die Logdateien enthalten den Verbrauch des jeweiligen Prozesses alle zwei Sekunden im CSV-Format (*Comma Seperated Values*). Diese Logdateien wurden mit Plotly[42] visualisiert.

Die Abbildungen 20, 21 und 22 zeigen die Systemauslastung in Hinsicht auf CPU- und Ram-Verbrauch beider Algorithmen für alle drei Datensätze.

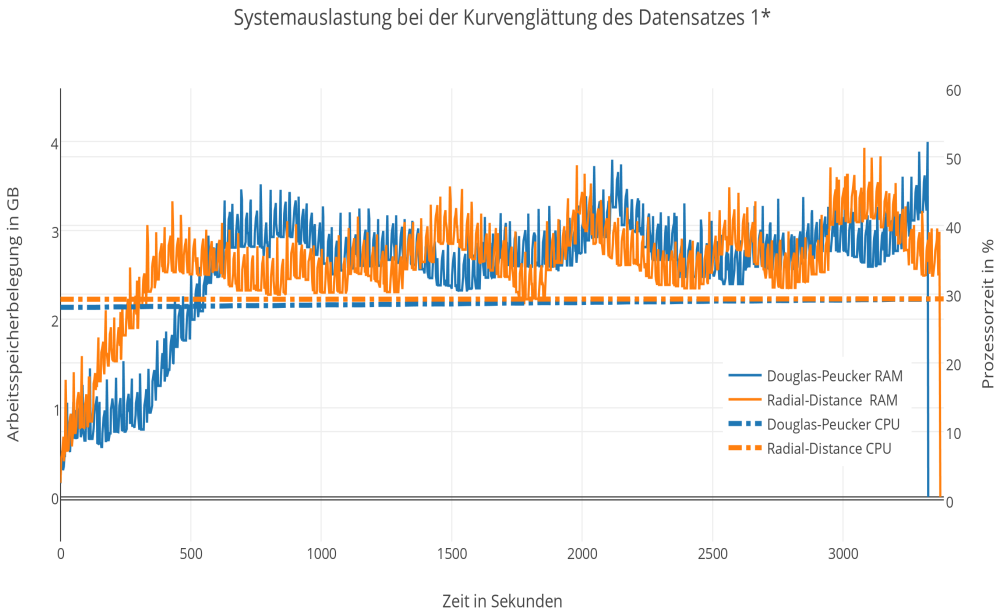


**Abbildung 20:** *Systemauslastung bei der Kurvenglättung des Datensatzes 1*

[42] http://plot.ly - Zugriff: 29.03.2015

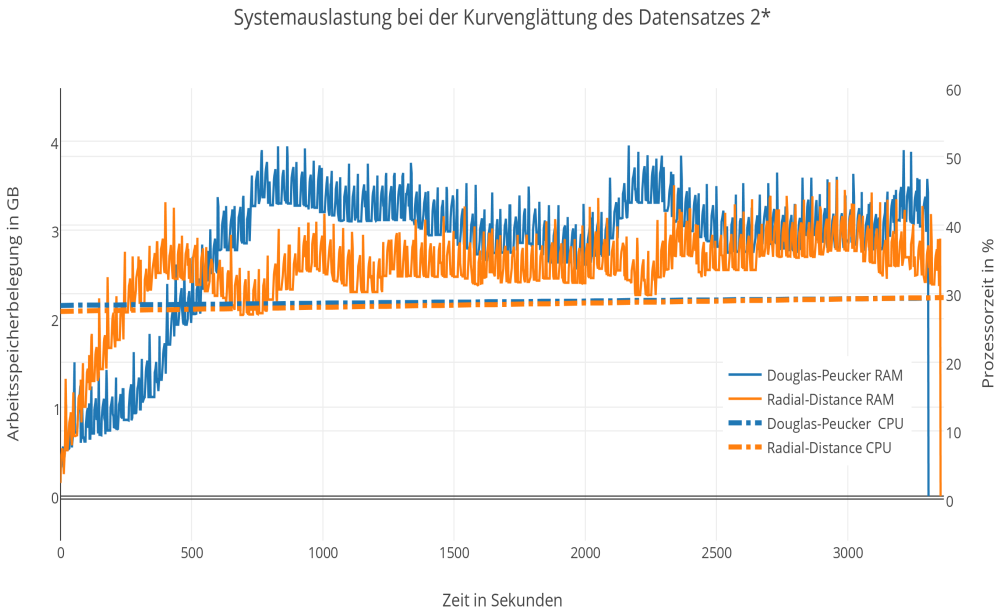


**Abbildung 21:** *Systemauslastung bei der Kurvenglättung des Datensatzes 2*

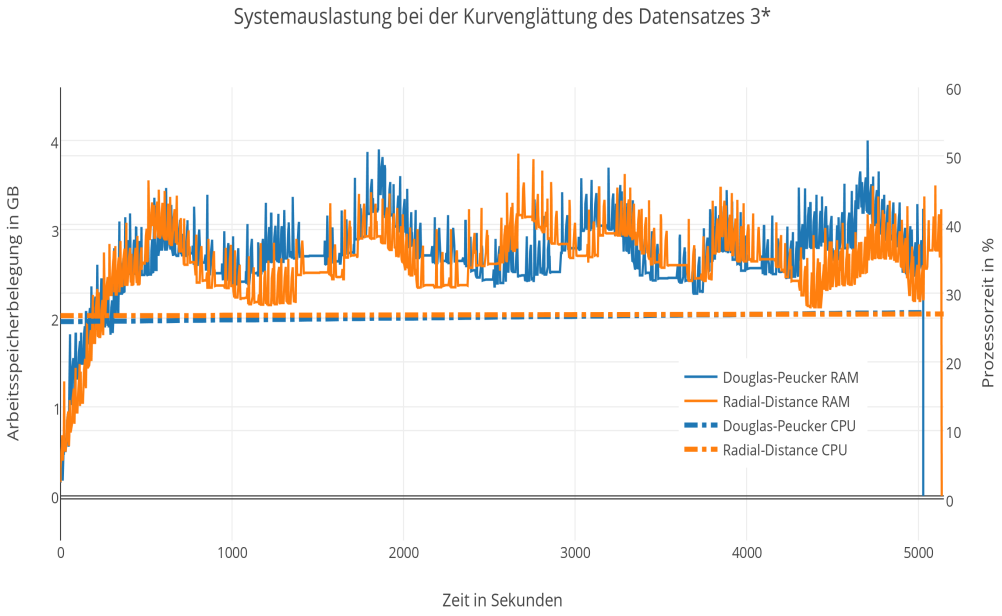


**Abbildung 22:** *Systemauslastung bei der Kurvenglättung des Datensatzes 3*

Die X-Achse zeigt die Anzahl der Sekunden, die seit Beginn des Programms, verstri-

chen sind. Die linke Y-Achse zeigt die Arbeitsspeicherbelegung durch die Prozesse java.exe, compare.exe und convert.exe, während der Laufzeit in Gigabyte an. Den Großteil des Arbeitsspeicherbedarfs machen die temporären Dateien von convert.exe aus. Die Konvertierung von Bildern der Größe 4950x4090 (Höhe und Breite) erfordert entsprechend große temporäre Dateien, da die Bilder unkomprimiert in den Arbeitsspeicher geladen werden. Javas *Garbage Collector* versäumt es die Objekte der heruntergeladenen Bilder zu löschen, was ebenfalls zu einer erhöhten Arbeitsspeicherbelegung führt. Die manuelle Überschreibung dieser Objekte mit *null* wäre eine mögliche Option diesen Umstand zu verbessern. Die rechte Y-Achse zeigt den Anteil der, für die jeweiligen Prozesse belegte Prozessorzeit an. Es war notwendig eine lineare Fitfunktion auf die Kurve der Prozessorzeit anzuwenden, da die Werte in kurzen Abständen zwischen geringen Werten wie >10% und hohen Werten wie <90% gesprungen sind. Die Fitfunktion: $f(x) = 28.28 \cdot x + 0.000324$, wobei $x$ die Prozessorzeit-Werte sind. Dadurch wird eine Repräsentation der Kurven ermöglicht, die eine übersichtlichere Visualisierung ermöglichen. Die extreme Schwankung der Prozesszeit liegt daran, dass sehr oft Ein-/Ausgabe-Operationen durchgeführt werden, um z.B. die Kacheln herunterzuladen. Ähnlich wie bei der Analyse der Ergebnisse zur Bildqualität führt die Analyse der Abbildungen 20, 21 und 22 zu der Folgerung, dass sich die beiden Algorithmen in ihrer Auswirkung auf das System nicht stark genug unterscheiden.

## 5.4 Diskussion

Nachdem die Ausgangsdaten (Kapitel 5.1), die Implementierung des Dienstes (Kapitel 5.2) und die Evaluierungsmetriken (Kapitel 5.3) vorgestellt wurden, folgt eine Diskussion, um die durch die vorliegende Bachelorarbeit erlangten Erkenntnisse in ihrer Gesamtheit zu erfassen. Die Diskussion ermöglicht auch eine abschließende Bewertung, ob und inwiefern ein Umwandlungsdienst, der WMS-Rasterdaten automatisch zu GeoJSON-Vektordaten konvertiert, einer manuellen Umwandlung derselben Art zu bevorzugen ist.

**manuelle Umwandlung vs. automatischer Umwandlungsdienst:** In Kapitel 3 wurde kurz darauf eingegangen, wie zeitaufwändig und mühsam es ist, Rasterdaten manuell in entsprechende Vektordaten umzuwandeln. Ein Vorteil gegenüber der automatischen Umwandlung ist, die Freiheit über die Genauigkeit und Detailtreue der umgewandelten Daten. Was Fehler betrifft, können in beiden Varianten unvorhergesehene Fehler auftreten. Zum einen ist der Mensch dafür anfällig, Fehler

selbst bei gut eingeübten Tätigkeiten zu machen. Andererseits haben die Ergebnisse der Evaluierungsdaten gezeigt, dass es auch in der automatischen Umwandlung zu Fehlern kommen kann.
Des Weiteren hat dieser Umwandlungsdienst noch keine Möglichkeit, die Legende für die unterschiedlich gefärbten Polygone in die Vektordaten einzubinden. Deshalb muss immer die Legende, die der Umwandlungsdienst als ein Bild mitliefert betrachtet werden, um die Bedeutung der unterschiedlichen Farben zu verstehen. Es ist aber möglich die Legende in die Vektordatei einzubinden, indem der Umwandlungsdienst erweitert wird. Im Frontend könnte die Legende gezeigt werden und für jede Farbe ein Textfeld erscheinen, in das der Klient den entsprechenden Wert für eine Farbe einträgt.
Eine weitere Einschränkung für diesen Umwandlungsdienst sind kleine Symbole auf der Karte, die ein Objekt darstellen, wie z.B. ein rotes Kreuz auf einem weißen Quadrat für ein Krankenhaus. Der in dieser Bachelorarbeit entwickelte Umwandlungsdienst würde diese Symbole als Flächen auf der Karte erkennen und da sie höchst wahrscheinlich sehr klein sind, werden sie entfernt und gehen verloren. Dieses Problem kann der Umwandlungsdienst lösen, indem wieder der Klient zu Beginn im Frontend auf der angezeigten Legende die erwünschten Symbole markiert. Stoßt der Umwandlungsdienst auf dieses Symbol, könnte er die Koordinaten des Zentrums dieses Symbols berechnen und anstelle eines Polygons der GeoJSON-Datei einen Punkt mit den berechneten Koordinaten hinzufügen.
Wird der Umwandlungsdienst auf einem langsamen Rechner, oder einem Rechner mit wenig internem Speicher ausgeführt, sodass externe Speichergeräte benutzt werden müssen, dauert die Umwandlung sehr lange. Das liegt vor Allem daran, dass die Kacheln zu einer sehr großen Bilddatei zusammengefügt werden. Die dabei entstehende temporäre Datei kann mehr als 20 Gigabyte groß sein und somit die Arbeitsspeicherkapazität vieler Rechner übersteigen. Dadurch muss die temporäre Datei im internen Speicher und wenn dort ebenfalls zu wenig Platz vorhanden ist, in einem externen Speicher gelagert werden. Diese Umstände können die Laufzeit des Programms um mehrere Stunden verlängern. Um das zu verhindern, kann ein System benutzt werden, dem aktuelle Hardwarekomponenten zur Verfügung stehen, damit es nicht zu solchen Engpässen kommt.
Ein weiteres Problem, welches sich negativ auf die Laufzeit auswirkt, ist die Programmiersprache Java, in der der Umwandlungsdienst hauptsächlich programmiert wurde. Obwohl primitive Datentypen stets Objekten vorgezogen wurden, wenn es möglich war, bietet Java dennoch einen Nachteil, der in Javas Struktur und Funktionsweise steckt. Simple arithmetische Operationen sind mit Java zwar ähnlich schnell

durchzuführen wie mit maschinennäheren Sprachen, wie z.B. C++ oder Assembler, die Zugriffe auf Variablen oder Durchführung von Operationen sind aber langsamer, da in Java jedes mal überprüft wird, ob es erlaubt ist die vorliegende Operation auszuführen. Bei maschinennahen Programmiersprachen hingegen findet ein direkter Zugriff statt, was schneller geht.
Ein weiterer Nachteil dafür, dass der Umwandlungsdienst in Java geschrieben wurde, ist die Tatsache, dass die verwendeten Bibliotheken wie Gdal/Ogr und ImageMagick lediglich "Wrapper" sind und die installierten Programme aufrufen. Diese Programme sind in c++ geschrieben und es wäre eine bessere Einbindung dieser Programme möglich.
Letzendlich kann dieser Umwandlungsdienst aber dafür verwendet werden, um viele Rasterdatensätze, die die Berliner Senatsverwaltung anbietet in Vektordaten umzuwandeln. Dadurch können viele Kosten und viel Zeit gespart werden. Hinzu kommt, dass der Umwandlungsdienst noch erweitert werden kann und somit noch viel Potenzial hat.

# 6 Zusammenfassung und Ausblick

Aufgrund der Verbreitung von mobilen Geräten, wie Smartphones und Tablet-PCs wird das Vektormodell immer beliebter, wenn es darum geht, Geodaten digital zu speichern. Standards, die aus Rohdaten Rasterdaten herstellen, wie z.B. der *Web Map Service* verlieren daher immer mehr an Bedeutung. Daher ist es wichtig Rasterdaten, die unter anderem von WMS-Servern geliefert werden, in entsprechende Vektordaten umzuwandeln. Ein weiterer Vorteil von Vektordaten ist es, dass mit ihnen eine bessere maschinelle Verarbeitung und Analyse möglich ist. Das liegt daran, dass Vektordaten geographische Informationen über ihren Inhalt haben. Bei Bildern, die ein WMS-Server schickt, ist das nicht der Fall. Der im Rahmen dieser Bachelorarbeit entstandene Umwandlungsdienst zeigt, dass es möglich ist, den Prozess der Umwandlung von Rasterdaten in Vektordaten zu automatisieren. Hierzu können bereits vorhandene Lösungen zu verschiedenen Problemen benutzt werden, wie z.B. die Kantenglättung um die Auflösung von Bildern zu verringern.
Es gibt einige wichtige Faktoren für den Bau eines Umwandlungsdienstes. Die Wahl welche Formate unterstützt werden, ist eine davon. Dem Kapitel 3.2 ist entnehmbar, dass es verschiedene Formate zu dem Vektormodell gibt und die Vor- und Nachteile dieser berücksichtigt werden müssen. Je mehr Formate unterstützt werden, desto anfälliger wird der Dienst gegenüber Konvertierungsfehlern. Auch sollte darauf geachtet werden ein Format zu nehmen, das von möglichen externen Bibliotheken auf die der Umwandlungsdienst zugreift, unterstützt wird. Ein weiterer Faktor ist die Wahl des Koordinatensystems. Wird ein lokales Koordinatensystem gewählt, besteht die Gefahr, dass Rasterdaten von Orten, die das Koordinatensystem nicht abdeckt, fehlerhaft oder überhaupt nicht umgewandelt werden.
Durch ihre Struktur weisen Rasterdaten “Treppenstufen” auf, wo eigentlich Diagonale Linien sein müssten. Diese Stufen werden bei der Extraktion der Polygone übernommen und so müssen diese Polygone vereinfacht werden. Es gibt eine Vielzahl von Algorithmen, die das können, aber zum Einen muss ein Algorithmus gewählt werden, der hinreichend schnell ist und zum Anderen müssen die Ergebnisse aber gut sein. Die Schnelligkeit von Algorithmen kann relativ einfach gemessen werden, z.B. so wie in der Evaluierung in Kapitel 5. Die Qualität der Ergebnisse hingegen sind nicht so einfach zu messen, da es von der subjektiven Einschätzung des Individuums abhängt. Bei der Wahl der Toleranz muss ausprobiert und so entschieden werden, da es dafür keine Richtlinien und Formeln gibt. Das hängt von den einzelnen Datensätzen ab.
Der wichtigste Aspekt jedoch ist, die Serialisierung der Polygone. Damit ist gemeint,

dass die Polygone in den Bildern extrahiert werden, indem die Eckpunkte des Polygons ermittelt und die Koordinaten dieser berechnet werden.
Es muss außerdem noch darauf geachtet werden Sonderfälle und Fehler, die vom WMS-Server stammen, abzufangen. Die WMS-Server der Berliner Senatsverwaltung schicken z.B. die Grenzen der sogenannten "lebensorientierten Räume" , obwohl in der Anfrage spezifiziert wurde diese nicht mitzuschicken.

**Ausblick:** Auch wenn der Umwandlungsdienst die Rasterdaten der WMS-Server der Berliner Senatsverwaltung in Vektordaten umwandeln kann, gibt es noch einige Verbesserungsmöglichkeiten, sowie weitere mögliche zusätzliche Funktionen. Wie in der Diskussion im fünften Kapitel erwähnt wurde, muss die Legende extra betrachtet werden.
Denn der Umwandlungsdienst berücksichtigt zwar die Farben der Polygone, aber nicht die Bedeutung der Farben. Diese Funktionalität kann erreicht werden, indem der Umwandlungsdienst so verändert wird, dass der Klient nachdem er die Optionen gewählt hat, die Informationen aus der Legende hinzufügen muss. Dazu kann die Legende angezeigt und der Nutzer darum gebeten werden, den Farben aus der Legende eine Beschreibung zuzuordnen. Diese Beschreibung könnte dann in den Vektordaten gespeichert sein, sodass die Legende überflüssig wird.

Ein anderer Aspekt der verbessert werden kann, ist die Anzahl der unterstützten Koordinatenreferenzsysteme zu erhöhen. Zur Zeit wird nur das EPSG:3068 unterstützt und die Ausgabedaten sind in EPSG:4326. Das liegt daran, dass die Kacheln, die laut den Koordinaten benachbart sein müssten, nicht benachbart sind. Das ist ein Fehler, der durch den WMS-Server erzeugt wird. Diesem Problem könnte entgegengetreten werden, indem kleinere Kacheln angefordert werden. Das würde die Verschiebung der benachbarten Kacheln verringern.

Ein weiteres Problem dieses Umwandlungsdienstes ist sein hoher Ressourcen- und Zeitverbrauch. Die Ursache dafür liegt an der Art, wie der Umwandlungsdienst die Vektordaten erzeugt. Es teilt nämlich die gesamte Karte in hundert Kacheln auf, bearbeitet sie und fügt sie dann zu einem großen Bild zusammen. Vor allem der letzte Schritt sorgt für den hohen Ressourcen- und Zeitverbrauch, da die Gesamtbilder in der Regel 40900 Pixel hoch und 49500 Pixel lang sind. Besser wäre es, die Polygone aus den einzelnen Kacheln zu extrahieren und im Anschluss die Vektordaten zusammenzufügen. Dabei muss beachtet werden, dass Polygone fälschlicherweise geteilt werden können. Das geschieht wenn ein Polygon über mehrere Kacheln verläuft.

Das wurde versucht in diesem Umwandlungsdienst umzusetzen. Eine externe Bibliothek namens *Gdal/Ogr*, welches auch benutzt wird, um die Polygone zu extrahieren, hat eine Funktion, die *dissolve* heißt. Mit ihr kann ein Attribut gewählt werden und alle Polygone, die dieses Attribut haben, werden zusammengefügt. Allerdings steht für die vorliegenden Polygone nur das Attribut für die Farbe zur Verfügung. Dadurch würden alle Polygone, egal ob sie benachbart sind oder nicht, zusammengefügt werden.

Die Visualisierung des Umwandlungsdienstes kann ebenfalls um weitere Funktionen ergänzt werden. Im Katalog befinden sich mehrere Datensätze über die selbe Thematik, die aber Daten von unterschiedlichen Zeiten beinhalten. Es wäre denkbar einen Regler einzubauen, der beim Verschieben die Datensätze zu verschiedenen Jahreszahlen visualisiert. Damit wäre eine Funktionalität geschaffen, die die Entwicklung über die Zeit besser verdeutlicht.

# A Anhang 1

XML-Dokument als Antwort auf die Anfrage auf: http://fbinter.stadt-berlin.de/fb/wms/senstadt/MsozS_S2_arblos252007Vz?REQUEST=GetCapabilities&SERVICE=WMS&VERSION=1.1.1 (Zugriff: 03.12.2014)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE WMT_MS_Capabilities SYSTEM "http://schemas.opengis.net/wms/1.1.1/capabilities_1_1_1.dtd" [
<!ELEMENT VendorSpecificCapabilities (#PCDATA)>]>
<WMT_MS_Capabilities xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1.1">
  <Service>
    <Name>OGC:WMS</Name>
    <Title>Arbeitslose unter 25 Jahren 2007</Title>
    <Abstract>Arbeitslose unter 25 Jahren in Prozent der 15-25-jährigen Einwohnerinnen und Einwohner 2007 auf Ebene der
          Verkehrszellen (Status 2)(Monitoring Soziale Stadtentwicklung 2008)</Abstract>
    <KeywordList>
      <Keyword>Population distribution - demography</Keyword>
      <Keyword>Soziales</Keyword>
      <Keyword>Karten</Keyword>
      <Keyword>Geodaten</Keyword>
      <Keyword>Berlin</Keyword>
      <Keyword>Monitoring Soziale Stadtentwicklung</Keyword>
      <Keyword>2008</Keyword>
      <Keyword>Arbeitslose</Keyword>
      <Keyword>Verkehrszellen</Keyword>
    </KeywordList>
    <OnlineResource xlink:href="http://www.stadtentwicklung.berlin.de/geoinformation/fis-broker/" xlink:type="simple" />
    <ContactInformation>
      <ContactPersonPrimary>
        <ContactPerson />
        <ContactOrganization>Senatsverwaltung für Stadtentwicklung und Umwelt Berlin</ContactOrganization>
      </ContactPersonPrimary>
      <ContactPosition>pointOfContact</ContactPosition>
      <ContactAddress>
        <AddressType>postal</AddressType>
        <Address />
        <City />
        <StateOrProvince />
        <PostCode />
        <Country>Germany</Country>
      </ContactAddress>
      <ContactVoiceTelephone>+49-30-90139-5257</ContactVoiceTelephone>
      <ContactElectronicMailAddress>fisbroker@senstadtum.berlin.de</ContactElectronicMailAddress>
    </ContactInformation>
    <Fees>Nutzungsbedingungen: Für die Verwendung der Daten gelten folgende Nutzungsbestimmungen: http://www.
          stadtentwicklung.berlin.de/geoinformation/download/nutzIII.pdf - Der Quellenvermerk gemäß § 2 lautet "
          Geoportal Berlin / [Titel des Datensatzes]".</Fees>
    <AccessConstraints>Es gelten keine Bedingungen</AccessConstraints>
  </Service>
  <Capability>
    <Request>
      <GetCapabilities>
        <Format>application/vnd.ogc.wms_xml</Format>
        <DCPType>
          <HTTP>
            <Get>
              <OnlineResource xlink:type="simple" xlink:href="http://fbinter.stadt-berlin.de/fb/wms/senstadt/
                    MsozS_S2_arblos252007Vz?" />
            </Get>
            <Post>
              <OnlineResource xlink:type="simple" xlink:href="http://fbinter.stadt-berlin.de/fb/wms/senstadt/
                    MsozS_S2_arblos252007Vz?" />
            </Post>
          </HTTP>
        </DCPType>
      </GetCapabilities>
      <GetMap>
        <Format>image/jpeg</Format>
```

```
58        <Format>image/png</Format>
59        <DCPType>
60          <HTTP>
61            <Get>
62              <OnlineResource xlink:type="simple" xlink:href="http://fbinter.stadt-berlin.de/fb/wms/senstadt/
                      MsozS_S2_arblos252007Vz?" />
63            </Get>
64            <Post>
65              <OnlineResource xlink:type="simple" xlink:href="http://fbinter.stadt-berlin.de/fb/wms/senstadt/
                      MsozS_S2_arblos252007Vz?" />
66            </Post>
67          </HTTP>
68        </DCPType>
69      </GetMap>
70      <GetFeatureInfo>
71        <Format>text/html</Format>
72        <Format>application/vnd.ogc.gml</Format>
73        <DCPType>
74          <HTTP>
75            <Get>
76              <OnlineResource xlink:type="simple" xlink:href="http://fbinter.stadt-berlin.de/fb/wms/senstadt/
                      MsozS_S2_arblos252007Vz?" />
77            </Get>
78            <Post>
79              <OnlineResource xlink:type="simple" xlink:href="http://fbinter.stadt-berlin.de/fb/wms/senstadt/
                      MsozS_S2_arblos252007Vz?" />
80            </Post>
81          </HTTP>
82        </DCPType>
83      </GetFeatureInfo>
84    </Request>
85    <Exception>
86      <Format>application/vnd.ogc.se_xml</Format>
87    </Exception>
88    <VendorSpecificCapabilities />
89    <!-- Layers -->
90    <Layer queryable="0" opaque="0">
91      <Title>Arbeitslose unter 25 Jahren 2007</Title>
92      <Abstract>Arbeitslose unter 25 Jahren in Prozent der 15-25-jährigen Einwohnerinnen und Einwohner 2007 auf Ebene
            der Verkehrszellen (Status 2)(Monitoring Soziale Stadtentwicklung 2008)</Abstract>
93      <KeywordList>
94        <Keyword>Karten</Keyword>
95        <Keyword>Geodaten</Keyword>
96        <Keyword>Berlin</Keyword>
97        <Keyword>Monitoring Soziale Stadtentwicklung</Keyword>
98        <Keyword>2008</Keyword>
99        <Keyword>Arbeitslose</Keyword>
100       <Keyword>Verkehrszellen</Keyword>
101     </KeywordList>
102     <SRS>EPSG:3068</SRS>
103     <SRS>EPSG:4326</SRS>
104     <LatLonBoundingBox minx="13.0512" miny="52.3246" maxx="13.7785" maxy="52.6935" />
105     <BoundingBox minx="850" miny="-150" maxx="50350" maxy="40750" SRS="EPSG:3068" />
106     <AuthorityURL name="GDI-DE">
107       <OnlineResource xlink:type="simple" xlink:href="http://www.gdi-de.org" />
108     </AuthorityURL>
109     <Identifier authority="GDI-DE">7cd2cf00-7bf2-36e7-9536-e2348e7b65bc</Identifier>
110     <MetadataURL type="TC211">
111       <Format>text/xml</Format>
112       <OnlineResource xlink:href="http://fbinter.stadt-berlin.de/fb/csw?REQUEST=GetRecordById&amp;SERVICE=CSW&amp;
              VERSION=2.0.2&amp;ID=4890f62d-8b3b-3c44-883f-ffc41fafbc26&amp;ELEMENTSETNAME=FULL" />
113     </MetadataURL>
114     <Style>
115       <Name>default</Name>
116       <Title>Standard</Title>
117       <LegendURL width="72" height="72">
118         <Format>image/gif</Format>
119         <OnlineResource xlink:href="http://fbinter.stadt-berlin.de/fb_daten/legenden/sozstadt/2008/S2_Legende2007Vz.
                gif" />
120       </LegendURL>
121     </Style>
```

```
122       <Layer queryable="0" opaque="0">
123         <Name>0</Name>
124         <Title>Arbeitslose unter 25 Jahren 2007</Title>
125         <KeywordList>
126           <Keyword>Karten</Keyword>
127           <Keyword>Geodaten</Keyword>
128           <Keyword>Berlin</Keyword>
129           <Keyword>Monitoring Soziale Stadtentwicklung</Keyword>
130           <Keyword>2008</Keyword>
131           <Keyword>Arbeitslose</Keyword>
132           <Keyword>Verkehrszellen</Keyword>
133         </KeywordList>
134         <SRS>EPSG:3068</SRS>
135         <SRS>EPSG:4326</SRS>
136         <LatLonBoundingBox minx="13.0512" miny="52.3246" maxx="13.7785" maxy="52.6935" />
137         <BoundingBox minx="850" miny="-150" maxx="50350" maxy="40750" SRS="EPSG:3068" />
138         <BoundingBox minx="13.0512" miny="52.3246" maxx="13.7785" maxy="52.6935" SRS="EPSG:4326" />
139         <Identifier authority="GDI-DE">7cd2cf00-7bf2-36e7-9536-e2348e7b65bc</Identifier>
140         <MetadataURL type="TC211">
141           <Format>text/xml</Format>
142           <OnlineResource xlink:href="http://fbinter.stadt-berlin.de/fb/csw?REQUEST=GetRecordById&amp;SERVICE=CSW&amp;
                  VERSION=2.0.2&amp;ID=4890f62d-8b3b-3c44-883f-ffc41fafbc26&amp;ELEMENTSETNAME=FULL" />
143         </MetadataURL>
144         <Style>
145           <Name>gdi_default</Name>
146           <Title>Standard GDI Style</Title>
147           <LegendURL width="300" height="300">
148             <Format>image/png</Format>
149             <OnlineResource xlink:href="http://fbinter.stadt-berlin.de/fb/wms/legend/senstadt/MsozS_S2_arblos252007Vz?
                    layerID=0" />
150           </LegendURL>
151         </Style>
152         <ScaleHint min="0" max="105" />
153       </Layer>
154       <Layer queryable="0" opaque="0">
155         <Name>1</Name>
156         <Title>Hintergrund 1:5.000- 1:1000</Title>
157         <Abstract>digitale Karte 1:5.000, ALK 1:1000</Abstract>
158         <KeywordList>
159           <Keyword>Karten</Keyword>
160           <Keyword>Geodaten</Keyword>
161           <Keyword>Berlin</Keyword>
162           <Keyword>Monitoring Soziale Stadtentwicklung</Keyword>
163           <Keyword>2008</Keyword>
164           <Keyword>Arbeitslose</Keyword>
165           <Keyword>Verkehrszellen</Keyword>
166         </KeywordList>
167         <SRS>EPSG:3068</SRS>
168         <SRS>EPSG:4326</SRS>
169         <LatLonBoundingBox minx="13.0512" miny="52.3246" maxx="13.7785" maxy="52.6935" />
170         <BoundingBox minx="850" miny="-150" maxx="50350" maxy="40750" SRS="EPSG:3068" />
171         <BoundingBox minx="13.0512" miny="52.3246" maxx="13.7785" maxy="52.6935" SRS="EPSG:4326" />
172         <Identifier authority="GDI-DE">7cd2cf00-7bf2-36e7-9536-e2348e7b65bc</Identifier>
173         <MetadataURL type="TC211">
174           <Format>text/xml</Format>
175           <OnlineResource xlink:href="http://fbinter.stadt-berlin.de/fb/csw?REQUEST=GetRecordById&amp;SERVICE=CSW&amp;
                  VERSION=2.0.2&amp;ID=4890f62d-8b3b-3c44-883f-ffc41fafbc26&amp;ELEMENTSETNAME=FULL" />
176         </MetadataURL>
177         <Style>
178           <Name>gdi_default</Name>
179           <Title>Standard GDI Style</Title>
180           <LegendURL width="300" height="300">
181             <Format>image/png</Format>
182             <OnlineResource xlink:href="http://fbinter.stadt-berlin.de/fb/wms/legend/senstadt/MsozS_S2_arblos252007Vz?
                    layerID=1" />
183           </LegendURL>
184         </Style>
185         <ScaleHint min="0" max="105" />
186       </Layer>
187       <Layer queryable="0" opaque="0">
188         <Name>2</Name>
```

```
189         <Title>Verkehrszellennummern und Konturen</Title>
190         <KeywordList>
191           <Keyword>Karten</Keyword>
192           <Keyword>Geodaten</Keyword>
193           <Keyword>Berlin</Keyword>
194           <Keyword>Monitoring Soziale Stadtentwicklung</Keyword>
195           <Keyword>2008</Keyword>
196           <Keyword>Arbeitslose</Keyword>
197           <Keyword>Verkehrszellen</Keyword>
198         </KeywordList>
199         <SRS>EPSG:3068</SRS>
200         <SRS>EPSG:4326</SRS>
201         <LatLonBoundingBox minx="13.0512" miny="52.3246" maxx="13.7785" maxy="52.6935" />
202         <BoundingBox minx="850" miny="-150" maxx="50350" maxy="40750" SRS="EPSG:3068" />
203         <BoundingBox minx="13.0512" miny="52.3246" maxx="13.7785" maxy="52.6935" SRS="EPSG:4326" />
204         <Identifier authority="GDI-DE">7cd2cf00-7bf2-36e7-9536-e2348e7b65bc</Identifier>
205         <MetadataURL type="TC211">
206           <Format>text/xml</Format>
207           <OnlineResource xlink:href="http://fbinter.stadt-berlin.de/fb/csw?REQUEST=GetRecordById&amp;SERVICE=CSW&amp;
                VERSION=2.0.2&amp;ID=4890f62d-8b3b-3c44-883f-ffc41fafbc26&amp;ELEMENTSETNAME=FULL" />
208         </MetadataURL>
209         <Style>
210           <Name>gdi_default</Name>
211           <Title>Standard GDI Style</Title>
212           <LegendURL width="300" height="300">
213             <Format>image/png</Format>
214             <OnlineResource xlink:href="http://fbinter.stadt-berlin.de/fb/wms/legend/senstadt/MsozS_S2_arblos252007Vz?
                  layerID=2" />
215           </LegendURL>
216         </Style>
217         <ScaleHint min="0" max="105" />
218       </Layer>
219     </Layer>
220   </Capability>
221 </WMT_MS_Capabilities>
```

# Literatur

[wms, ] OpenGIS Web Map Service (WMS) Implementation Specification 1.3.0. Technical report.

[gml, 2008] (2008). Gml. In *Encyclopedia of GIS*, pages 409–409. Springer US.

[Aitchison, 2012] Aitchison, A. (2012). *Pro Spatial with SQL Server 2012*. Apress, Berkely, CA, USA, 1st edition.

[Basham et al., 2008] Basham, B., Sierra, K., and Bates, B. (2008). *Head first servlets and JSP - passing the Sun certified web component developer exam: a brain-friendly guide: new mock exam included (2. ed.)*. O'Reilly.

[Boutell, 1997] Boutell, T. (1997). PNG (Portable Network Graphics) Specification Version 1.0. RFC 2083 (Informational).

[Braden, 1986] Braden, B. (1986). The Surveyor's Area Formula. *The College Mathematics Journal*, 17:326–337.

[Burdziej, 2012] Burdziej, J. (2012). A web-based spatial decision support system for accessibility analysis—concepts and methods. *Applied Geomatics*, 4(2):75–84.

[Chandler and Hyatt, 2002] Chandler, K. and Hyatt, K. (2002). *Customer-centered Design: A New Approach to Web Usability*. Prentice Hall Press, Upper Saddle River, NJ, USA, first edition.

[Dahlström et al., 2011] Dahlström, E., Dengler, P., Grasso, A., Lilley, C., McCormack, C., Schepers, D., Watt, J., Ferraiolo, J., Jun, and Jackson, D. (2011). Scalable Vector Graphics (SVG) 1.1 (Second Edition).

[Davis, 2007] Davis, S. (2007). *GIS for Web Developers: Adding Where to Your Web Applications*. Pragmatic Bookshelf, Raleigh, NC.

[de Lange, 2007] de Lange, N. (2007). *Geoinformatik: in Theorie und Praxis*. Springer London, Limited.

[Decker, 2001] Decker, D. (2001). *GIS Data Sources*. Wiley.

[DeMers, 2009] DeMers, M. (2009). *GIS For Dummies*. –For dummies. Wiley.

[Di, 2008] Di, L. (2008). Standards, Critical Evaluation of Remote Sensing. In Shekhar, S. and Xiong, H., editors, *Encyclopedia of GIS*, pages 1128–1135. Springer US.

[Douglas and Peucker, 1973] Douglas, D. H. and Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122.

[Edney, 1996] Edney, M. H. (1996). Theory and the History of Cartography. *Imago Mundi*, 48:185–191.

[Feeman, 2008] Feeman, T. G. (2008). Mathematical foundations of gis. In *Encyclopedia of GIS*, pages 637–651. Springer US.

[Fling, 2009] Fling, B. (2009). *Mobile Design and Development: Practical Concepts and Techniques for Creating Mobile Sites and Web Apps - Animal Guide*. O'Reilly Media, Inc., 1st edition.

[Furht, 2006a] Furht, B. (2006a). Jpeg. In Furht, B., editor, *Encyclopedia of Multimedia*, pages 372–374. Springer US.

[Furht, 2006b] Furht, B. (2006b). Portable network graphics (png). In Furht, B., editor, *Encyclopedia of Multimedia*, pages 707–708. Springer US.

[Heywood et al., 2006] Heywood, D., Cornelius, S., and Carver, S. (2006). *An Introduction to Geographical Information Systems*. Pearson educación. Pearson Prentice Hall.

[Hoel, 2008] Hoel, E. G. (2008). Data models in commercial gis systems. In Shekhar, S. and Xiong, H., editors, *Encyclopedia of GIS*, pages 215–219. Springer.

[Kedlaya, 2006] Kedlaya, K. S. (2006). Geometry unbound.

[Kralidis, 2008] Kralidis, A. T. (2008). Geospatial Open Source and Open Standards Convergences. In Hall, G. and Leahy, M., editors, *Open Source Approaches in Spatial Data Handling*, volume 2 of *Advances in Geographic Information Science*, pages 1–20. Springer Berlin Heidelberg.

[Krygier and Wood, 2011] Krygier, J. and Wood, D. (2011). *Making Maps, Second Edition: A Visual Guide to Map Design for GIS*. Guilford Publications.

[Lee et al., 2005] Lee, E., Kim, M., Kim, M., and Joo, I. (2005). A web services framework for integrated geospatial coverage data. In *Computational Science and Its Applications–ICCSA 2005*, pages 1136–1145. Springer Berlin Heidelberg.

[Michaelis and Ames, 2008] Michaelis, C. D. and Ames, D. P. (2008). Web Feature Service (WFS) and Web Map Service (WMS). In *Encyclopedia of GIS*, pages 1259–1261. Springer US.

[Mokrzycki and Samko, 2012] Mokrzycki, W. and Samko, M. (2012). Canny edge detection algorithm modification. In Bolc, L., Tadeusiewicz, R., Chmielewski, L., and Wojciechowski, K., editors, *Computer Vision and Graphics*, volume 7594 of *Lecture Notes in Computer Science*, pages 533–540. Springer Berlin Heidelberg.

[Moretz, 2008] Moretz, D. (2008). Internet gis. In *Encyclopedia of GIS*, pages 591–596. Springer US.

[Morris, 2010] Morris, S. (2010). Preservation of geospatial data: the connection with open standards development. In Jobst, M., editor, *Preservation in Digital Cartography*, Lecture Notes in Geoinformation and Cartography, pages 129–146. Springer Berlin Heidelberg.

[Murray, 2013] Murray, S. (2013). *Interactive Data Visualization for the Web*. O'Reilly Media, Inc.

[Open Geospatial Consortium, 2007] Open Geospatial Consortium (2007). Schema for coverage geometry and functions. Technical Report OGC 07-011.

[Open Geospatial Consortium, 2010] Open Geospatial Consortium (2010). *OpenGIS Web Coverage Service version 2.0*. Number OGC 09-110r3 in OpenGIS Standard. Open Geospatial Consortium (OGC).

[Open Geospatial Consortium, 2012] Open Geospatial Consortium (2012). *OpenGIS Geography Markup Language version 3.3*. Number OGC 10-129r1 in OpenGIS Standard. Open Geospatial Consortium (OGC).

[Percivall, 2008a] Percivall, G. (2008a). OGC Reference Model (OGC 08-062r4). Technical report.

[Percivall, 2008b] Percivall, G. (2008b). OGC's Open Standards for Geospatial Interoperability. In *Encyclopedia of GIS*, pages 800–805. Springer US.

[Percivall, 2010] Percivall, G. (2010). Progress in OGC Web Services Interoperability Development. In Di, L. and Ramapriyan, H. K., editors, *Standard-Based Data and Information Systems for Earth Observation*, Lecture Notes in Geoinformation and Cartography, pages 37–61. Springer Berlin Heidelberg.

[Pretzsch, 2009] Pretzsch, H. (2009). *Forest dynamics, growth and yield. From measurement to model*. Springer, Berlin, Heidelberg.

[Rayner and Schmidt, 1969] Rayner, W. and Schmidt, M. (1969). *Fundamentals of Surveying*. Van Nostrand-Reinhold.

[Rice and Knight, 1973] Rice, H. and Knight, R. (1973). *Technical mathematics*. McGraw-Hill.

[Richard Groot, 2000] Richard Groot, J. D. M. (2000). *Geospatial Data Infrastructure*. Oxford University Press.

[Roelofs, 1999] Roelofs, G. (1999). *PNG - the definitive guide: creating and programming portable network graphics*. O'Reilly.

[van Kreveld et al., 1997] van Kreveld, M. J., Nievergelt, J., Roos, T., and Widmayer, P., editors (1997). *Algorithmic Foundations of Geographic Information Systems, this book originated from the CISM Advanced School on the Algorithmic Foundations of Geographic Information Systems, Udine, Italy, September 16-20, 1996*, volume 1340 of *Lecture Notes in Computer Science*. Springer.

[Vretanos, 2005] Vretanos, P. A. (2005). Web feature service implementation specification. Technical report, OGC.

[Woll, 1992] Woll, A., editor (1992). *Wirtschaftslexikon*. Oldenbourg, München [u.a.], 6., überarb. und erw. aufl. edition.

[Woodward and Harley, 2007] Woodward, D. and Harley, J. B. (2007). *Vol. 3. The history of cartography*. Univ. of Chicago Press, Chicago [u.a.].

[Yue et al., 2010] Yue, P., Di, L., Zhao, P., Yang, W., Yu, G., and Wei, Y. (2010). Semantic Augmentations to an ebRIM Profile of Catalogue Service for the Web. In Di, L. and Ramapriyan, H. K., editors, *Standard-Based Data and Information Systems for Earth Observation*, Lecture Notes in Geoinformation and Cartography, pages 189–208. Springer Berlin Heidelberg.