Master Thesis

im Rahmen des

Universitätslehrganges „Geographical Information Science & Systems“

(UNIGIS MSc) am Zentrum für Geoinformatik (Z_GIS)

der Paris Lodron-Universität Salzburg

zum Thema

„Visualisierung von Straßenbefahrungsdaten in einem WebGIS“

Performanceorientierte Optimierung der Rasterdatenhaltung

Vorgelegt von

Dipl. Geogr. Stefan Wächter
U1391, UNIGIS MSc Jahrgang 2008

Zur Erlangung des Grades

„Master of Science (Geographical Information Science & Systems) – MSc(GIS)“

Gutachter:

Ao. Univ. Prof. Dr. Josef Strobl

Tengling, 28.03.2011

Erklärung der eigenständigen Abfassung der Arbeit

"Ich versichere, diese Master Thesis ohne fremde Hilfe und ohne Verwendung anderer als der angeführten Quellen angefertigt zu haben, und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat. Alle Ausführungen der Arbeit die wörtlich oder sinngemäß übernommen wurden sind entsprechend gekennzeichnet."

Tengling, 28.03.2011 _____

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

AGG	Anti-Grain Geometry
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CCITT	International Telegraph and Telephone Consultative Committee
CGI	Common Gateway Interface
CSS	Cascading Style Sheets
DCT	Diskreten Kosinus-Transformation
DOM	Document Object Model
DOP	Digitales Orthophoto
FGSV	Forschungsgesellschaft für Straßen- und Verkehrswesen e.V.
FOSS4G	Free and Open Source Software for Geomatics
GD	graphics draw
GDAL	Geospatial Data Abstraction Library
GEOS	Geometry Engine Open Source
GIF	Graphics Interchange Format
HTTP	Hypertext Transfer Protocol
GML	Geography Markup Language
GWT	Google Web Toolkit
IFD	Image File Directory
IFH	Image File Header
INS	Inertiales Navigationssystem
JFIF	JPEG File Interchange Format
JSON	JavaScript Object Notation
KVP	keyword-value pair
MNDNR	Minnesota Department of Natural Resources
MS4W	MapServer for Windows
OGC	Open Geospatial Consortium
OSGeo	OpenSource Geospatial Foundation
OSGi	Open Standard Gateway Interface
REST	Representational State Transfer
SLD	Styled Layer Descriptor
OWS	OGC Web Service
SOS	Sensor Observation Service

TIFF	Tagged Image File Format
JPEG	Joint Photographic Experts Group
TMS	Tile Map Service
UMN	University of Minnesota
URL	Uniform Resource Locator
VRT	Virtual Dataset
W3C	World Wide Web Consortium
WCS	Web Coverage Service
WFS	Web Feature Service
WFS-T	Web Feature Service transactional
WMS	Web Map Service
WMTS	Web Map Tile Service
XHTML	Hypertext Markup Language
XML	Extensible Markup Language

Kurzfassung

Die vorliegende Arbeit befasst sich mit der Visualisierung von Straßenbefahrungsdaten in einem WebGIS. Aus der Befahrung resultieren hochaufgelöste Digitale Orthophotos, die dem Anwender zur Verfügung gestellt werden sollen. Aus diesem Grund ist eine Optimierung der Rasterdatenhaltung notwendig.

Ziel der Arbeit ist Konfiguration eines Kartendienstes, der, im Hinblick auf die Performance, optimale Ergebnisse bei der Visualisierung von Rasterdaten erzielt.

Es werden die theoretischen Grundlagen onlinebasierter Geoinformationstechnologien erörtert, um in einem nächsten Schritt den Aufbau einer WebGIS Infrastruktur mit ihren Komponenten zu skizzieren.

Zur Wahl des Datenformates für die zu verarbeitenden Rasterdaten werden Tests, die sich am MapServer UMN WMS Benchmark der Free and Open Source Software for Geomatics Conference (FOSS4G) orientieren, durchgeführt. In einer virtuellen Testumgebung werden die Ausgangsdaten (TIFF) in die Formate TILED TIFF, PNG, GIF, JPEG und JPEG2000 konvertiert, um sie schließlich dem Benchmark unterziehen zu können. Dabei werden auch die zur Verfügung stehenden Schnittstellen Common Gateway Interface (CGI) und FastCGI sowie die Ausgabeformate von MapServer UMN berücksichtigt. Zum Vergleich werden auch andere Rasterdaten herangezogen.

Auf Grundlage der Testergebnisse werden die Daten als Dienst zur Verfügung gestellt und in ein WebGIS Portal integriert. Verwendung findet dabei das mapAccel 3.0 WebGIS Applikationframework. Die Konfigurationsschritte zur Einrichtung des WebGIS Clients sowie die zu verwendende Systemarchitektur werden ausführlich beschrieben.

Abstract

This paper deals with the visualization of road survey data in a Web-based GIS. Users shall be provided with high resolution orthophotographs that were taken during the survey. Due to this fact the management of raster data has to be optimized.

Intention of the thesis is the configuration of a map service which, with regard to the performance, achieves optimal results in visualization of raster data.

There is a discussion of the basic principles of online-based geoinformation technologies to outline the configuration of a WebGIS infrastructure with its components.

In order to choose the proper image file format, tests are performed, which are geared to the annual MapServer UMN WMS Benchmark of Free and Open Source Software for Geomatics Conference (FOSS4G). In a virtual testing environment the original datasets (TIFF) are converted to tiled TIFF, PNG, GIF, JPEG and JPEG2000 in order to run the benchmark. The influence of MapServer UMN output format on the performance is measured, as well as Common Gateway Interface (CGI) and FastCGI which were used to interact with the WMS server. Moreover other raster datasets were tested to compare the results.

Based on these findings, the data are served as a service and are integrated into a WebGIS portal. The mapAccel 3.0 WebGIS Application Framework was used to tackle this task. The configuration of the WebGIS client and the system architecture is described step by step.

1 Einführung

Das überörtliche Straßennetz der Bundesrepublik Deutschland (Autobahnen, Bundes-, Landes und Kreisstraßen) besaß im Jahr 2010 eine Länge von etwa 231000 Kilometern und stellt damit einen der wichtigsten Verkehrswege dar (vgl. STATISTISCHES BUNDESAMT, 2010a). 78 Prozent der Beförderungsmenge im Güterverkehr entfielen 2009 auf die Straße (vgl. STATISTISCHES BUNDESAMT, 2010b). Diese Zahlen unterstreichen deren volkswirtschaftliche Bedeutung. Wenn man sich vor Augen führt, dass dieses Netz ständigen Degradations-, Instandsetzungs- und Erneuerungsprozessen unterworfen ist, wird sofort klar, wie wichtig ein zeitgemäßes Informationsmanagement in diesem Bereich ist.

Es steht außer Frage, dass sich Geographische Informationssysteme für diese Art der Datenverarbeitung und -bereitstellung eignen. Auch hier schreitet die Entwicklung besonders auf dem Gebiet der Internet-Kartendienste voran. Wohl am häufigsten in der Literatur angeführt, ist der Start von Google Maps im Jahr 2005, der nicht nur einen Meilenstein in technologischer Hinsicht, sondern auch in Bezug auf die Akzeptanz von Web-Mapping Applikationen für eine breite Öffentlichkeit darstellt. Als Gründe für diesen Erfolg können auch die Geschwindigkeit beim Kartenaufbau und die intuitive Benutzerführung ausgemacht werden. Dies ist, mehr denn je, bei der Entwicklung neuer Anwendungen zu berücksichtigen. Die Informationen können somit nicht nur Spezialisten, sondern auch fachübergreifend einem großen Anwenderspektrum zur Verfügung gestellt werden.

1.1 Ausgangslage und Problemstellung

Durch die Reform des kommunalen Haushalts- und Rechnungswesens in Deutschland soll das System der doppelten Buchführung (Doppik) die Kameralistik ablösen. Die Konzepte der einzelnen Bundesländer werden dabei unterschiedlich bezeichnet (vgl. PROMBERGER et al., 2004). Die Verkehrsinfrastruktur bildet ein nicht unerhebliches kommunales Vermögen, das für die Eröffnungsbilanz bewertet werden muss. Nicht allen Kommunen liegen diese Informationen in geeigneter Form vor.

Die Bestandsdaten- und Zustandserfassung kann entweder visuell durch geschultes Personal oder mit Hilfe messtechnischer Verfahren durchgeführt werden (vgl. FGSV, 2003). Technologieführend in diesem Bereich ist das *eagle eye* Konzept von eagle eye technologies GmbH. Es handelt sich dabei um ein kinematisches, terrestrisches, photo-

grammetrisches Aufnahme- und Scansystem, das mit direkter Georeferenzierung arbeitet. Ein Messfahrzeug, das mit einem Inertialen Navigationssystem (INS), GPS und odometrischen Sensoren ausgestattet ist, erfasst mit schiefachsig zueinander montierten Digitalkameras die Fahrbahn. Optional erfolgt die Aufnahme zusätzlich mit einem Laserscanner.

Folgende Abbildung zeigt das Multi-Sensor-System:

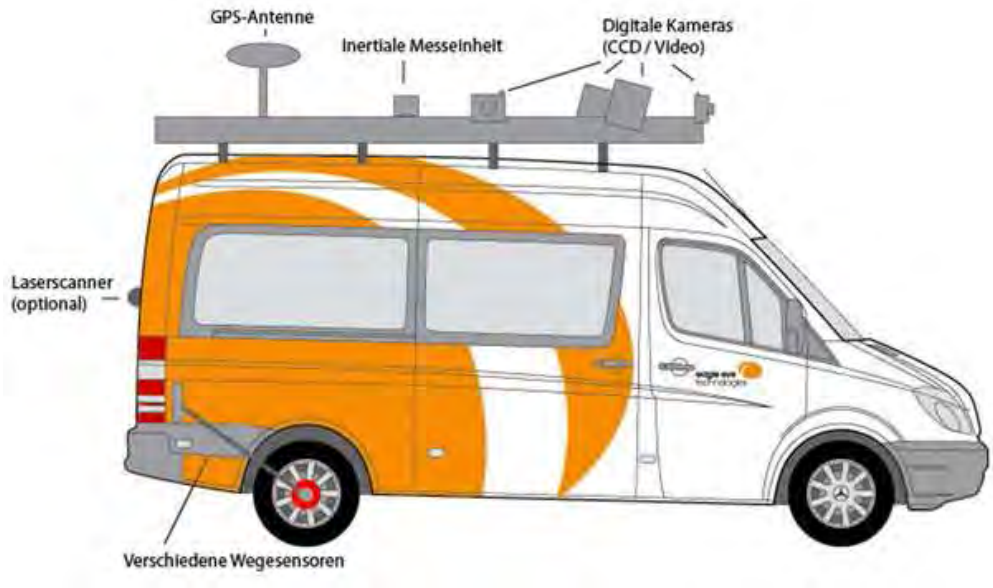


Abb. 1: Messfahrzeug eagle eye technologies (http://www.ee-t.de, 2010)

Neben diesen passpunktfreien Bilddaten können mit Hilfe von senkrecht zur Fahrbahn ausgerichteten Digitalkameras, extrem hochaufgelöste, georeferenzierte Orthophotos der Straßendecke aufgenommen werden. Auflösungen unter 1mm sind möglich. Folgende Abbildung zeigt ein Orthophoto mit 2mm Bodenauflösung. Es ist nahezu jedes Detail des Schachtdeckels erkennbar:

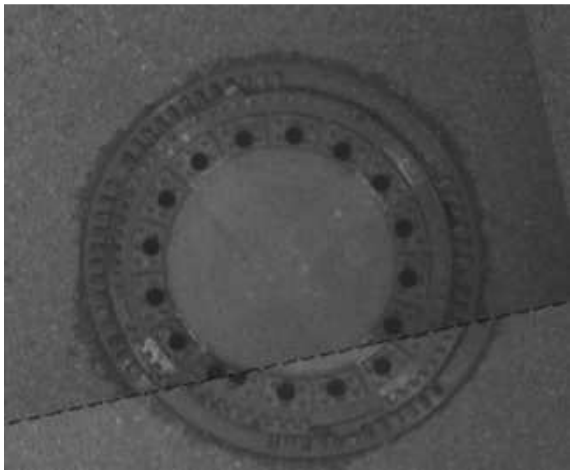


Abb. 2: hochaufgelöstes Orthophoto (Quelle: eagle eye technologies GmbH)

Im darauffolgenden Postprocessing werden die Stereoaufnahmen photogrammetrisch ausgewertet und die Straßenzustände erfasst (vgl. BÄUMKER und LUDWIG, 2007).

Die Visualisierung dieser Ergebnisse soll in einem webbasierten Geographischen Informationssystem erfolgen. Dabei nimmt die Performance des Kartendienstes eine zentrale Rolle ein. Die Geschwindigkeit des Systems ist vor allem für die Anwenderfreundlichkeit und die Akzeptanz von großer Bedeutung. Die Möglichkeit der nachhaltigen Nutzung dieser Daten, Offenheit und die Integration in eine (kommunale) Geodateninfrastruktur, stellt eine weitere Anforderung an das System dar.

1.2 Lösungsansatz und Zielsetzung

Das Ziel der vorliegenden Arbeit ist es, eine optimale Konfiguration eines Kartendienstes im Hinblick auf die performante Verarbeitung von Rasterdaten zu erarbeiten und damit die mit dem *eagle eye* Verfahren erhobenen Daten zu visualisieren. Dabei gilt es zu erörtern, wie die möglichen Ausgangsformate die Geschwindigkeit des Kartendienstes beeinflussen und ob die Ausgabeformate für die Fragestellung bedeutsam sind. Aufgrund der Interoperabilitätsanforderungen an das System, werden auch gängige Standards für die Auslieferung von Rasterkarten betrachtet. Welche in Frage kommen und dabei implementiert werden können, muss dargelegt werden. Die Untersuchung beschränkt sich dabei auf *MapServer UMN*, dem Kartenserver, der vermutlich den weltweit größten Verbreitungsgrad besitzt (vgl. KORDUAN / ZEHNER, 2008). Als WebGIS Lösung wird das *mapAccel* Framework von Territorium Online GmbH verwendet.
Wenngleich die Datengrundlage und die Anforderungen aus dem Verkehrsinfrastrukturbereich stammen, stellt das Erhaltungsmanagement nicht den Untersuchungsgegenstand der vorliegenden Arbeit dar. Auch die fahrzeuggestützte Objekterfassung wird nicht näher behandelt.

1.3 Struktur dieser Thesis

Die Arbeit beginnt mit einem kurzen Überblick über die verwendete Literatur und einem Abriss über den Stand der Forschung im Bereich WebGIS.

Im darauffolgenden Abschnitt werden die für die Arbeit essentiellen Techniken und die Grundlagen onlinebasierter Geoinformationstechnologien erläutert. Darunter fällt zum einen eine Einführung in die verwendete Client / Server Architektur, die Formulierung von Standards, als auch eine Beschreibung der gängigsten Grafikformate.

Kapitel 5 beschreibt den Aufbau und die Durchführung des Benchmarks zur Ermittlung der optimalen Serverkonfiguration. Die Ergebnisse werden vorgestellt und analysiert.

Im Anschluss daran wird der Aufbau einer WebGIS Infrastruktur beschrieben und die Konfiguration des mapAccel WebGIS Clients und Applikationsservers anhand der Straßenbefahrungsdaten umgesetzt.

Zum Schluss der Arbeit werden die Ergebnisse zusammengefasst und kritisch diskutiert. Ein Ausblick über künftige Entwicklungen schließt diese Thesis ab. Folgende Abbildung veranschaulicht die Struktur dieser Thesis:

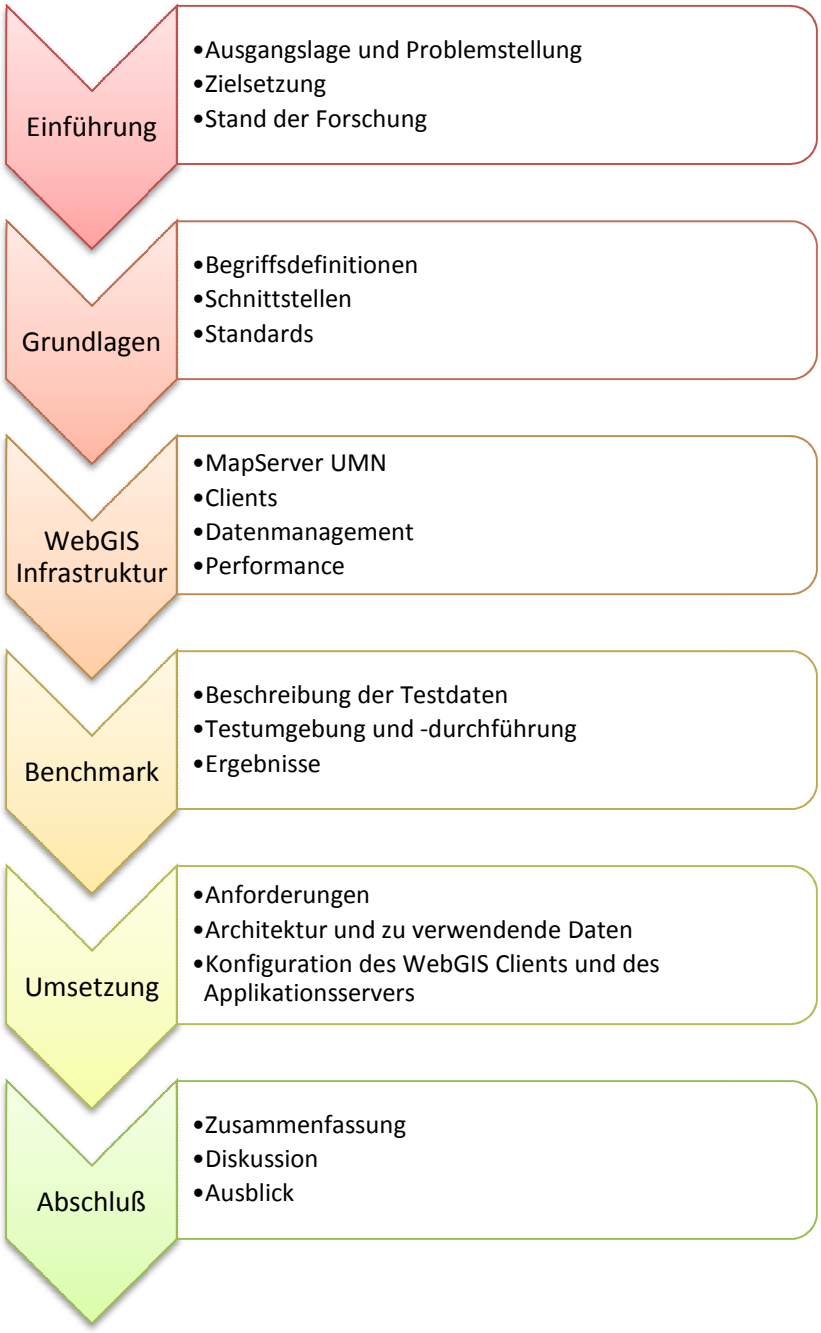


Abb. 3: Struktur der Master Thesis (eigener Entwurf)

2 Stand der Forschung

2.1 Literaturüberblick

Einen umfassenden Überblick zur Thematik „Internet-GIS“, liefern KORDUAN und ZEHNER (2008). Es werden sowohl die informationstechnischen Grundlagen als auch gängige Standards dargelegt. Dem Leser wird ein grundlegender Einblick in die Technologien und Möglichkeiten, die WebGIS bietet, ermöglicht.

Die Herausgeber HALL und LEAHY (2008) bündeln in ihrer Veröffentlichung die wichtigsten Entwicklungen im Bereich Open Source und Open Standards im Bereich der Geoinformationsverarbeitung. Für die vorliegende Arbeit sind vor allem die Artikel von LIME zu MapServer UMN, WARMERDAM zur Geospatial Data Abstraction Library und CHEN und XiE zu Open Source Datenbanken und deren räumliche Erweiterungen von Bedeutung.

Die Veröffentlichung von KROPLA (2005) stellt, neben der Onlinedokumentation, das Standardwerk zur Konfiguration von MapServer UMN dar.

SAMPLE und IOUP (2010) widmen sich ausführlich dem Thema Tiling. Die Technologie und die zugrundeliegenden Prämissen werden detailliert vorgestellt. Darüber hinaus wird ebenso die Erzeugung, die Speicherung und der praktische Einsatz von gekachelten Rasterdaten beschrieben. Fallstudien ergänzen und veranschaulichen die theoretischen Ausführungen.

Da für die vorliegende Arbeit hauptsächlich Rasterdaten als Datengrundlage dienen, wurden mehrere Werke herangezogen. Dazu zählen MURRAY und VANRYPER (1996), die mit ihrer ENCYLOPEDIA OF GRAPHICS FILE FORMATS eines der Standardwerke zum Thema Grafikformate bilden. BURGER und BURGE (2009) liefern die Grundlagen der digitalen Bildverarbeitung. STRUTZ (2009) beschreibt detailliert die waveletbasierte Bildkompression als auch die Standards zur Einzelbildkompression.

Die Empfehlungen für das Erhaltungsmanagement von Innerortsstraßen, E EMI 2003, der Forschungsgesellschaft für Straßen- und Verkehrswesen bilden den theoretischen Hintergrund für die in dieser Arbeit zu visualisierenden Verkehrsinfrastrukturdaten. Die Technik des *eagle eye* Verfahrens wird ausführlich in BÄUMKER und LUDWIG (2007) beschrieben. Für die fahrzeuggestützte Objekterfassung wird auf HARING (2007) verwiesen.

Dedizierte Untersuchungen zur Performance von MapServer UMN hat ZETTEL (2007) durchgeführt. Allerdings beschränkt er sich in seiner Untersuchung rein auf Vektordaten.

2.2 Theoretische Grundlagen

Die mit den ständig wachsenden Geodatenbeständen gestiegenen Anforderungen an Informationsaustausch zwischen den Systemen und den Akteuren im Internet, machten die Einführung von einheitlichen Standards und Schnittstellen unabdingbar. Die Fähigkeit heterogene Datenquellen nutzen zu können und komponentenunabhängig zu kommunizieren wird in der Informationstechnologie unter dem Begriff Interoperabilität subsumiert (vgl. KORDUAN und ZEHNER, 2008). Sie stellt die Grundlage für Geodateninfrastrukturen dar.

Einer der wichtigsten Standards wurde 1999 vom Open Geospatial Consortium (OGC) verabschiedet – der *Web Map Service (WMS)*. Dieser Dienst liefert Geodaten als Raster in Form von Grafikformaten aus. WMS Dienste von unterschiedlichen Quellen lassen sich kombinieren und als neuer Dienst einbinden. Der *Web Coverage Service (WCS)* stellt multidimensionale Rasterkarten bzw. Grids zur Verfügung. Im Gegensatz zu WMS, werden beim WCS auf dem Server keine Grafiken gerendert und an den Client gesendet, sondern die Daten mit ihrer Semantik direkt übergeben, um sie weiter verarbeiten zu können – anstelle einer reinen Visualisierung (vgl. OGC, 2010). Der *Web Feature Service (WFS)* ermöglicht es Anwendungen auf Vektordaten im *Geography Markup Language (GML)* Format, einem weiteren OGC Standard, zuzugreifen. Vereinfacht kann er als Pendant zum WCS gesehen werden. Im Jahr 2010 verabschiedete die OGC eine Spezifikation für den *Web Map Tile Service (WMTS)*, der den WMS Dienst ergänzt. WMTS standardisiert die Auslieferung von gekachelten (tiled) Rasterkarten. Als Vorlage diente dafür der im organisatorischen Rahmen der *Open Source Geospatial Foundation (OSGeo)* entstandene *Tile Map Service (TMS)*, der allerdings nicht als offizieller Standard anzusehen ist (vgl. OCG, 2010 und http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification).

Bei letzterem ist die Verbindung von *Open Standard* und *Open Source* erkennbar. Unter Open Standard versteht man Standards, die öffentlich verwendet werden können. Open Standards können zwar in Open Source Software integriert werden, stehen dieser aber nicht exklusiv zur Verfügung (vgl. KRALIDIS, 2008). Der Trend weist in Richtung Open Source.

Die OSGeo ist eine non-profit Organisation, die sich zum Ziel gesetzt hat, freie und Open Source Software im Bereich der Geoinformation zu unterstützen. Sie bildet ein administratives Umfeld für eine Reihe bekannter Projekte, wie z.B. OpenLayers, Mapguide, Quantum GIS oder GDAL/OGR. Im Gegensatz zur OGC verabschiedet die OSGeo keine Standards, sondern trägt zu deren Verbreitung bei (vgl. http://www.osgeo.org).

MCIHAGGA (2008) greift diese Thematik auf und liefert drei zentrale Gründe warum im Bereich „Web Mapping“ Open Source Projekte derart erfolgreich sind:

- Aufgrund des Nischencharakters der Absatzmärkte können proprietäre Produkte nicht betriebswirtschaftlich entwickelt werden
- Dienstleistungen werden stärker gewichtet als Produkte
- Vertikale Märkte erzeugen Anreize, um proprietäre Lösungen zu entwickeln, in die Open Source Technologien einfließen

Darüber hinaus existiert eine Reihe von „inoffiziellen“ Standards, die aufgrund ihrer Popularität als de facto Spezifikationen anzusehen sind. Eines der bekanntesten ist wohl das ESRI *Shape* Format. Für den Bereich des Internet-GIS ist *GeoJSON* zu nennen – das auf der *JavaScript Object Notation (JSON)* basierte Austauschformat für Geodaten (vgl. KRALIDIS, 2008).

Die genannten Standards und Spezifikationen sind die essentielle Grundlage für die Interoperabilität und leisten einen wichtigen Beitrag zum Aufbau von Geodateninfrastukturen. Nicht zu vergessen ist die *Extensible Markup Language (XML),* die vom World Wide Web Consortium (W3C) definiert wurde und die für einen Großteil der angeführten Standards als Auszeichnungssprache dient. Als weiterer Trend für die Internet-GIS Community ist der Einsatz von *AJAX* (Asynchronous JavaScript and XML) Technologie anzuführen, bei der der Browser nur diejenigen Bereiche vom Server anfordert, die auch wirklich neu dargestellt werden müssen. Damit lassen sich sehr performante und ansprechende Internet-GIS Lösungen realisieren.

Die hier kurz skizzierten Spezifikationen werden im folgenden Kapitel vertieft und um die für das Internet-GIS grundlegenden Technologien ergänzt.

3 Grundlagen onlinebasierter Geoinformationstechnologien

3.1 Begriffsdefinitionen und essentielle Techniken

Zu Beginn der theoretischen Ausführungen sollen nach der Definition von KORDUAN und ZEHNER (2008) die für onlinebasierte Geoinformationstechnologien verwendeten Begriffe anhand ihrer Funktionalität abgegrenzt werden.

Unter *Web-Mapping* fasst man Applikationen zusammen, die auswählbare Layer vorhalten und einfache Basisfunktionen, wie Zoomen oder das Verschieben des Kartenausschnitts, ermöglichen. Von *Web-GIS* Anwendungen wird gesprochen, wenn die Datenhaltung in einer Sachdatenbank erfolgt, die vom Nutzer abgefragt werden kann. Darüber hinaus stehen dem Anwender umfangreichere Analysefunktionen, z.B. Buffer, zur Verfügung. Im Folgenden wird der Begriff WebGIS dafür verwendet. Die funktionale Steigerung dessen, wird als *Internet-GIS* bezeichnet.

Da Webapplikationen, die den Funktionsumfang eines vollständigen Desktop GIS besitzen, noch eine Seltenheit darstellen, wird im weiteren Verlauf der Arbeit nur noch der Begriff WebGIS verwendet.

Im einfachsten Fall ist die Architektur einer WebGIS Anwendung im World Wide Web (WWW) oder im lokalen Netzwerk eines Unternehmens nach dem Client-Server Prinzip aufgebaut. Der Server nimmt eine Anfrage (Request) des Client entgegen und sendet eine Antwort (Response) zurück.

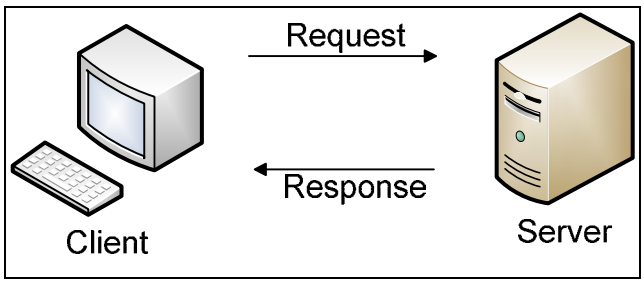


Abb. 4: Client-Server Architektur (eigener Entwurf)

Es gibt eine Vielzahl von Client-Server Kombinationen. Man kann zum einen den Client isoliert betrachten, zum anderen kann dem Client, im Normalfall ein Browser, vom Server Applikationslogik in Form von HTML oder Skriptsprachen zur Verfügung gestellt werden, sobald eine Anfrage an den Server gesendet wird. Browser Plug-ins, festinstallierte Clientsoftware, wie z.B. der SVG Viewer oder Flash Player von Adobe,

stellen einen weiteren Fall eines Clients dar. Eine Übersicht liefern KORDUAN und ZEHNER (2008).

Moderne Web-Applikationen basieren auf der Mehrschichten-Architektur. Da viele Clientzugriffe den Server und dessen Datenbank schnell überlasten könnten, versucht man die einzelnen Systeme zu verteilen. Für ein WebGIS bietet es sich an, dass der Client nur die Karten präsentiert und die Interaktion zwischen dem Anwender und dem Kartendienst übernimmt. Wie zuvor beschrieben, ist in diesem Fall die Applikationslogik in eine Webseite eingebettet. Diese Schicht wird als *Presentation-Tier* bezeichnet. Die Business Logik Schicht (*Business Logic Tier*) übernimmt die Ablaufsteuerung und verteilt die Daten, die im *Data Tier* verwaltet werden (vgl. SIMON et al., 2004).

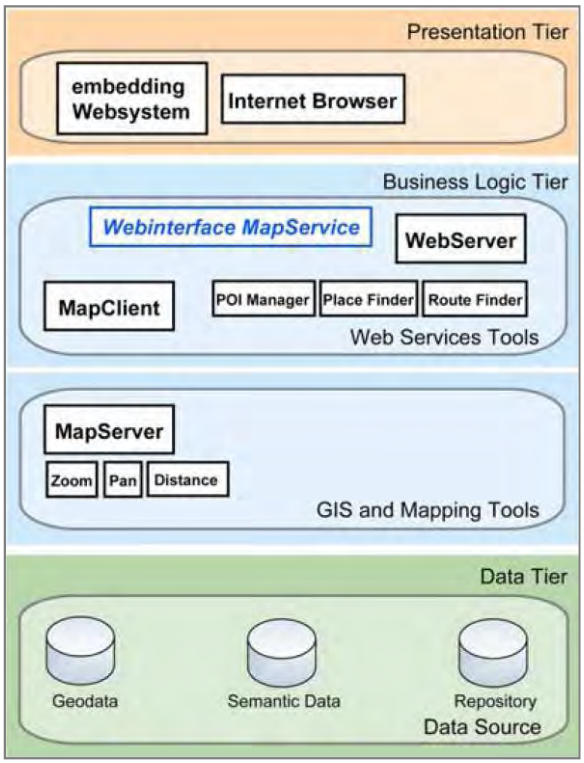


Abb. 5: Drei-Schichten-Architektur kartengestützter Onlinesysteme (SIMON et al., 2004)

Ruft der Client eine Webseite über deren *Uniform Resource Locator (URL)* auf, so kommuniziert er mit dem Server bzw. Kartendienst über das Hyper Text Transfer Protocol (HTTP) oder die verschlüsselte Variante HTTPS. Der Webserver wird die Karte in der Regel nicht selbst erzeugen, sondern lässt sie sich von einem Kartendienst, z.B. MapServer UMN, generieren. Dieser wiederum wird über seine HTTP-Client-Schnittstelle oder aber über eine serviceorientierte Schnittstelle angesprochen (vgl. SIMON et al., 2004). Der Kartendienst kommuniziert mit den (verteilten) Datenquellen per

TCP/IP-Datenübertragungsprotokoll. Nach dem Rendern der Karte, wird diese an den Client zurückgesendet. Diese kann dann in eine HTML Seite eingebettet sein.

In einer zweischichtigen Architektur, auch als Two-Tier-Modell bezeichnet, findet die Verteilung nur auf Client und Server statt. Diese Variante ist zwar leicht zu konfigurieren, allerdings nicht leicht zu erweitern (vgl. LI et al., 2002).

3.2 Schnittstellen

3.2.1 Common Gateway Interface (CGI)

Die Kommunikation zwischen Client und Server erfolgt über Schnittstellen. Eine der gängigsten ist dabei das *Common Gateway Interface (CGI)*. Über diese Schnittstelle können externe Programme serverseitig ausgeführt werden. Der Client kann dem CGI-Programm Parameter übergeben. Diese werden mit der HTTP GET oder der POST Methode an den Server übergeben. Bei der GET Methode werden die Parameter direkt an die URL gehängt und vom Webserver in Umgebungsvariablen geschrieben, die dem CGI-Programm zur Verfügung stehen. Bei der POST Methode werden die Parameter in die Standardeingabe (stdin) geschrieben und vom CGI-Programm ausgelesen. In der folgenden Abbildung veranschaulichen KORDUAN und ZEHNER (2008) den Ablauf eines CGI-Programms:

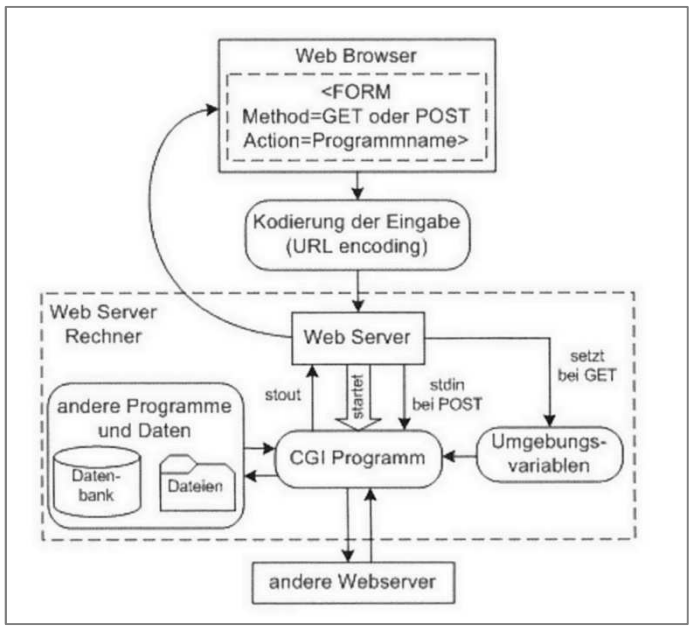


Abb. 6: Ablauf eines CGI-Programms (KORDUAN und ZEHNER, 2008 nach BENGEL, 2002)

Ein Nachteil der CGI-Schnittstelle ist die Tatsache, dass für jede Anfrage ein eigener Prozess auf Betriebssystemebene angestoßen wird, der unter Umständen ressourcenintensiv ist (vgl. SIMON et al., 2004).

3.2.2 Application Programming Interface (API)

Ein *Application Programming Interface (API)* ist im Gegensatz zur CGI-Schnittstelle wesentlich stärker serverorientiert (vgl. BARTELME, 2005). Proprietäre Server-APIs erlauben es, Programme direkt in Server einzubinden. Dies bringt eine Performancesteigerung im Gegensatz zur CGI-Schnittstelle. Eine weitere Kommunikationsmöglichkeit zwischen Server und Kartendienst bietet eine Servlet-API. Ein Servlet ist eine javabasierte Applikation, die auf einem Server in einem sogenannten Container läuft und auf Clientanfragen wartet. Beim Aufruf eines Servlets wird eine Java Klasse instanziiert und das Ergebnis an den Client zurückgesendet. Ein bekanntes Beispiel für einen Servlet Container ist der Apache Tomcat (http://tomcat.apache.org/).

3.2.3 OGC Schnittstellen

OGC Schnittstellen für bestimmte Dienste, z.B. WMS oder WFS, sind ähnlich aufgebaut. Auf die Anfrage eines Clients folgt die Antwort des Servers. Anfragen können, wie üblich, über HTTP mit der GET oder der POST Methode erfolgen. Ein Beispiel für einen GetCapabilities Aufruf mit der Methode GET und OGC-konformen Parametern könnte wie folgt aussehen:

```
http://labs.gatewaygeomatics.com/cgi-
bin/benchmark2010?SERVICE=WMS&VERSION=1.1.1&REQUEST=GetCap
abilities
```

Antwort dieser Anfrage ist ein XML Dokument, das Metadaten über den Dienst liefert. Ausführlich werden die Schnittstellen des WMS-Dienstes, z.B. GetMap, unter 3.3.1 beschrieben.

3.3 OGC Webservices

Von zentraler Bedeutung für die Umsetzung interoperabler Lösungen sind die vom OpenGIS Consortium veröffentlichten Standards für Webservices. Damit lassen sich nicht nur Daten über definierte Web-Schnittstellen austauschen, sondern es können damit auch Redundanzen in der Datenhaltung vermieden werden. Mittlerweile hat sich eine Vielzahl an OGC Webservices etabliert. Es werden jedoch in den folgenden Abschnitten nur diejenigen näher vorgestellt, die für die Beantwortung der Fragestellung

von Bedeutung sind. Dazu zählt in erster Linie der Web Map Service (WMS) und der Web Feature Service (WFS), der optional für Vektordaten eingesetzt werden kann. Für alle anderen sei auf die entsprechenden Kapitel in KORDUAN und ZEHNER (2008) oder auf die Homepage des OpenGIS Consortiums (http://www.opengeospatial.org/standards) verwiesen.

3.3.1 Web Map Service

Ein Web Map Service (WMS) liefert georeferenzierte Karten in der Form gängiger Bildformate oder auch vektorbasiert im Scalable Vector Graphics (SVG) Format aus. Die folgenden Ausführungen beziehen sich auf die aktuellste OpenGIS Web Map Service Implementation Specification in der Version 1.3.0 (OGC, 2006).

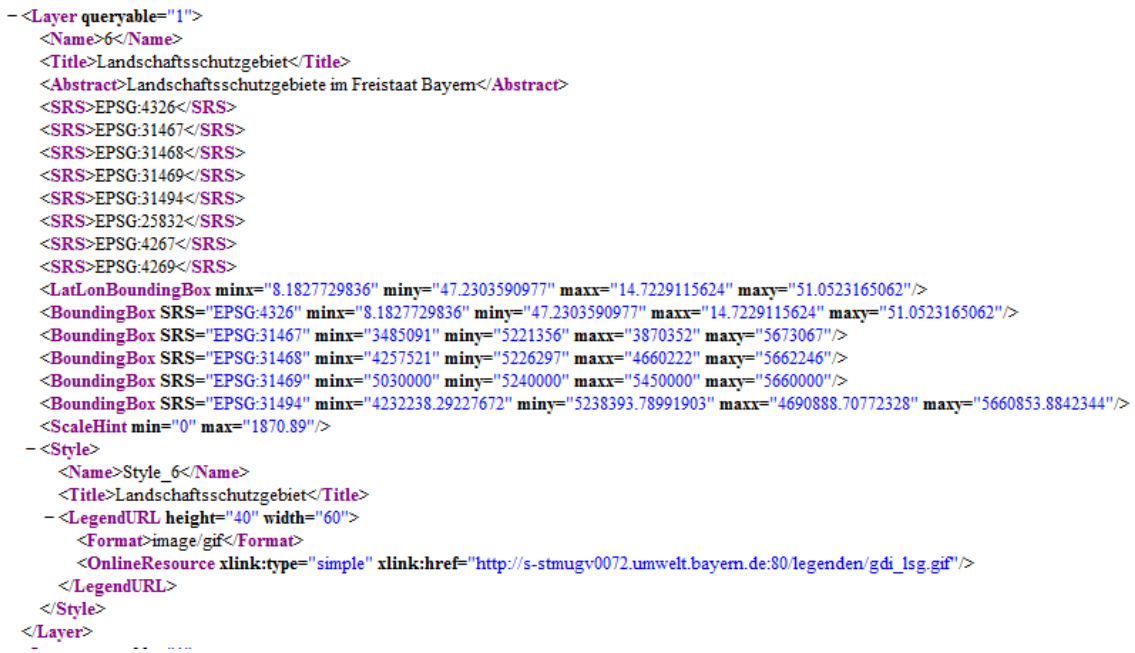
Der WMS Standard legt drei Operationen fest. *GetCapabilities* liefert Metadaten auf Serviceebene. *GetMap* gibt eine Karte aufgrund bestimmter Parameter zurück und die optionale *GetFeatureInfo* Operation liefert Informationen zu bestimmten Objekten auf der Karte aus. WMS Operationen werden in Form einer URL aufgerufen.

Zwingende Parameter eines GetCapabilities Aufrufs sind "Service=WMS" und "Request=GetCapabilities". Als Antwort erhält der Client ein XML-Dokument in dem die Service-Metadaten und die Capabilities enthalten sind. Ein Beispielaufruf könnte wie folgt aussehen:

http://irgendeineip/irgendeinewebapp?SERVICE=WMS&REQUEST=GetCapabilities

Die Metadaten über den Dienst enthalten dessen Namen und Titel, ein Abstract, Kontaktinformationen, die Ressourcen URL, Schlüsselworte sowie Informationen über Nutzungsbeschränkungen und Kosten (vgl. KORDUAN und ZEHNER, 2008). Die Capabilities beschreiben die Leistungen des Dienstes. Mögliche Operationen werden im Request Tag und Layerinformationen im Layer Tag aufgeführt. Beispielhaft ist in der folgenden Abbildung eine Layer Sektion des WMS Dienstes „Naturschutzgebiete Bayerns“ aufgeführt:

http://s-stmugv0072.umwelt.bayern.de/arcwms/com.esri.wms.Esrimap?Service=WMS&Request=GetCapabilities



```
- <Layer queryable="1">
    <Name>6</Name>
    <Title>Landschaftsschutzgebiet</Title>
    <Abstract>Landschaftsschutzgebiete im Freistaat Bayern</Abstract>
    <SRS>EPSG:4326</SRS>
    <SRS>EPSG:31467</SRS>
    <SRS>EPSG:31468</SRS>
    <SRS>EPSG:31469</SRS>
    <SRS>EPSG:31494</SRS>
    <SRS>EPSG:25832</SRS>
    <SRS>EPSG:4267</SRS>
    <SRS>EPSG:4269</SRS>
    <LatLonBoundingBox minx="8.1827729836" miny="47.2303590977" maxx="14.7229115624" maxy="51.0523165062"/>
    <BoundingBox SRS="EPSG:4326" minx="8.1827729836" miny="47.2303590977" maxx="14.7229115624" maxy="51.0523165062"/>
    <BoundingBox SRS="EPSG:31467" minx="3485091" miny="5221356" maxx="3870352" maxy="5673067"/>
    <BoundingBox SRS="EPSG:31468" minx="4257521" miny="5226297" maxx="4660222" maxy="5662246"/>
    <BoundingBox SRS="EPSG:31469" minx="5030000" miny="5240000" maxx="5450000" maxy="5660000"/>
    <BoundingBox SRS="EPSG:31494" minx="4232238.29227672" miny="5238393.78991903" maxx="4690888.70772328" maxy="5660853.8842344"/>
    <ScaleHint min="0" max="1870.89"/>
  - <Style>
      <Name>Style_6</Name>
      <Title>Landschaftsschutzgebiet</Title>
    - <LegendURL height="40" width="60">
        <Format>image/gif</Format>
        <OnlineResource xlink:type="simple" xlink:href="http://s-stmugv0072.umwelt.bayern.de:80/legenden/gdi_lsg.gif"/>
      </LegendURL>
    </Style>
  </Layer>
```


Abb. 7: Teil eines GetCapabilities Dokuments (verändert nach BAYERISCHES STAATSMINISTERIUM FÜR UMWELT UND GESUNDHEIT)

Mit Hilfe der GetMap Operation kann eine Karte in Form eines Bildes angefordert werden. Zwingend sind die in der folgenden Tabelle angeführten Parameter:

Parameter	Beschreibung
VERSION=1.3.0	Request Version
REQUEST=GetMap	Request Name
LAYERS	kommagetrennte Liste von Layern
STYLES	kommagetrennte Liste von Styles
CRS=namespace:identifier	räumliches Bezugssystem
BBOX=minx,miny,maxx,maxy	Bounding Box Koordinaten
WIDTH	Breite der Karte in Pixel
HEIGHT	Höhe der Karte in Pixel
FORMAT	Ausgabeformat der Karte

Tab. 1: verpflichtende Request Parameter der GetMap Operation (vgl. OGC, 2006)

Mit Karten unterschiedlicher Quellen, die die gleichen CRS, BBOX, WIDTH und HEIGHT Parameter besitzen, können Overlays erstellt werden, wenn der überlagernde Layer mit dem Parameter TRANSPARENT=TRUE angefordert wird. Dies gilt auch für Daten, die benachbart sind. Daraus ergibt sich die Möglichkeit einen WMS Server zu-

gleich als WMS Client einzusetzen und damit wiederum mehrere WMS Server einzubinden. Diese Konstellation wird als kaskadierter WMS Dienst bezeichnet (vgl. KORDUAN und ZEHNER, 2008).

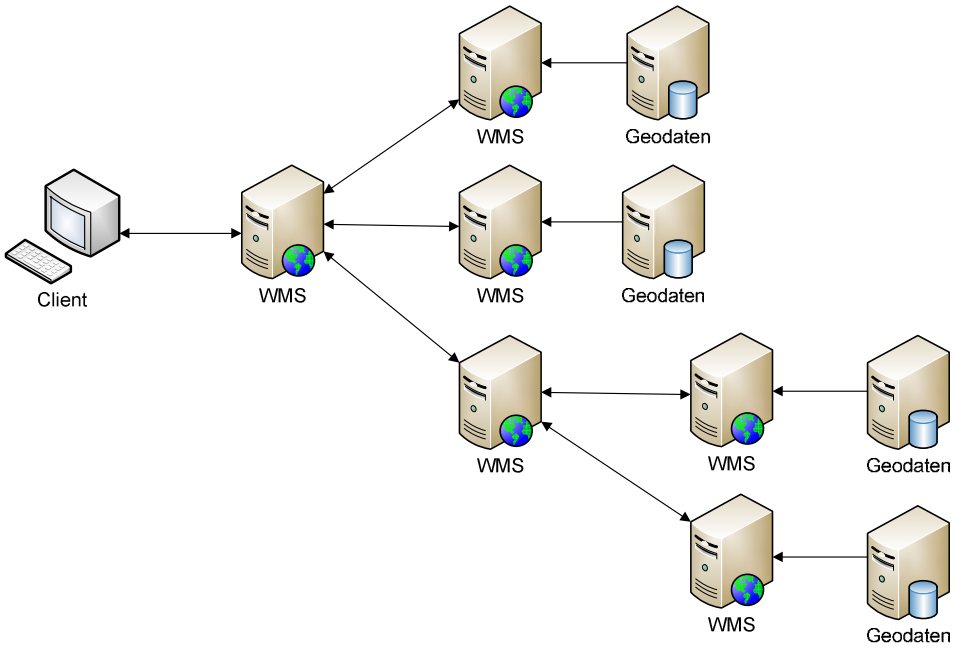


Abb. 8: kaskadierte WMS Dienste (eigener Entwurf)

Die optionale GetFeatureInfo Operation bietet die Möglichkeit, Sachdaten von WMS Layern abzufragen. Die verpflichteten Parameter sind in der folgenden Tabelle aufgeführt:

Parameter	**Beschreibung**
VERSION=1.3.0	Request Version
REQUEST=GetFeatureInfo	Request Name
map request part	Teil des GetMap Requests des abzufragenden Layers
QUERY_LAYERS	kommagetrennte Liste der abzufragenden Layer
INFO_FORMAT	MIME-Type der Antwort, z.B. text/html
I	Rechtswert in Pixel des Kartenkoordinatensystems, nach rechts ansteigend
J	Hochwert in Pixel des Kartenkoordinatensystems, nach unten ansteigend

Tab. 2: verpflichtende GetFeatureInfo Parameter (vgl. OGC, 2010)

Eine Erweiterung des WMS Dienstes stellt der *Styled Layer Descriptor (SLD)* dar. Mit ihm können Clients benutzerdefinierte Styles an den WMS Server weiterreichen, der diese dann beim Rendern der Karte berücksichtigt. In der Version 1.1.0 können zwei zusätzliche Operationen ausgeführt werden - *DescribeLayer* und *GetLegendGraphic* (vgl. OGC, 2007).

Bei einer GetMap Operation hat der Client drei Möglichkeiten SLD zu nutzen. Bei der ersten Möglichkeit kann der Client einen URL-Aufruf direkt auf das SLD Dokument per HTTP GET und dem Parameter SLD machen. Bei der zweiten bzw. dritten Variante kann der Client per HTTP GET oder HTTP POST den Parameter SLD_BODY übergeben (vgl. KORDUAN und ZEHNER, 2008).

3.3.2 Web Feature Service

Der Web Feature Service (WFS) ermöglicht lesenden und schreibenden Zugriff auf Vektordaten. Die folgenden Ausführungen beziehen sich auf die OGC Web Feature Service Implementation Specification in der Version 1.1.0. Die Schnittstelle ist, wie beim WMS, HTTP basiert, d.h. die Anfragen können mit GET oder POST erfolgen. Der Client kann die Eigenschaften des Dienstes mit GetCapabilities und optional auch Feature Definitionen abrufen. Wenn der Client einen Request an den WFS Server absetzt, sendet dieser zusätzlich zur Anforderung einen Statusreport an den Client nachdem der Request verarbeitet wurde. Der WFS unterstützt sechs verschiedene Operationen. Diese sind in nachfolgender Tabelle zusammengefasst:

Operation	**Beschreibung**
GetCapabilities	Beschreibung der Capabilities des Dienstes, z.B. zu liefernde Featuretypen und die Operationen, die auf diesen ausgeführt werden können
DescribeFeatureType	Beschreibung der Featuretypen
GetFeature	Auslieferung von Features
GetGmlObject	Zugriff auf GML Dokumente über XLinks
Transaction	schreibender Zugriff: Erzeugen, Aktualisieren oder Löschen von Daten
LockFeature	Sperren der Features für die Dauer der Transaktion

Tab. 3: WFS Operationen (vgl. OGC, 2005)

Auf Grundlage dieser Operationen können drei Klassen von WFS unterschieden werden:

Klasse	Beschreibung
Basic WFS	nur lesender Zugriff auf die Daten mit den Operationen GetCapabilities, DescribeFeatureType und GetFeature
XLink WFS	Unterstützt alle Operationen des Basic WMS plus GetGmlObjekt
Transaction WFS	Lese- und Schreibzugriff auf die Daten; unterstützt alle Operationen des Basic WMS plus GetGmlObjekt und LockFeature

Tab. 4: WFS Klassen (vgl. OGC, 2005):

Zwei Möglichkeiten bestehen, um Anfragen an den WFS Server zu stellen: Zum einen XML-kodiert mit der HTTP POST-Methode, zum anderen mit den HTTP GET-Parametern, die auch als keyword-value pairs (KVP) bezeichnet werden (vgl. OGC, 2005).

Für WFS Dienste gibt es drei Definitionen für Namensräume. Für das WFS Schnittstellen Vokabular wird http://www.opengeospatial.net/wfs, für das GML Vokabular http://www.opengeospatial.net/gml und für das OGC Filter Vokabular http://www.opengeospatial.net/ogc verwendet (vgl. OGC, 2005).

Wie von den anderen OGC Web Services (OWS) bereits bekannt, wird auch beim WFS Dienst mit der Operation GetCapabilities ein XML Dokument mit den Eigenschaften des Dienstes geliefert. Folgende Abschnitte enthält das Dokument:

Sektion	Beschreibung
Service Identification	Informationen über den WFS Dienst
Service Provider	Informationen über den Anbieter des WFS Dienstes
Operation Metadata	Informationen über die möglichen Operationen, deren Parameter und Bedingungen
FeatureType Liste	Liste aller Featuretypen und Operationen, die auf diese angewendet werden können; zusätzlich wird das Raumbezugssystem angegeben
ServesGMLObjectType Liste	Liste der GML Objekttypen, die von einem WFS Dienst stammen, der die GetGMLObject Operation unterstützt

	und nicht von gml:AbstractFeatureType abgeleitet sind
SupportsGMLObjectType Liste	„Liste aller GML Objekttypen, die einem GML-Schema, welches der WFS unterstützt, entsprechen.“ (KORDUAN und ZEHNER, 2008)
Filter Capabilities	Optionaler Abschnitt, der die Operationen enthält, die in der Filter Encoding Implementation Specification definiert sind

Tab. 5: WFS Capabilities Dokument (vgl. OGC, 2005)

Für WFS Requests sind die Parameter VERSION, SERVICE=WFS und REQUEST= verpflichtend. Die Operation DescribeFeatureType informiert den Client über die einzelnen vom WFS Dienst angebotenen Featuretypen.

Mit der GetFeature Operation werden die vom Client angeforderten Daten ausgeliefert. Eine Beispielabfrage zeigt die nachstehende Abbildung. Dabei wird eine Instanz des Featuretyps *InwaterA_1M* angefordert, die die ID *InWaterA_1M.1234* besitzt:

```
<?xml version="1.0" ?>
<wfs:GetFeature
    service="WFS"
    version="1.1.0"
    outputFormat="text/xml; subtype=gml/3.1.1"
    xmlns:myns="http://www.someserver.com/myns"
    xmlns:wfs="http://www.opengis.net/wfs"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/wfs ../wfs/1.1.0/WFS.xsd">
    <wfs:Query typeName="myns:InWaterA_1M">
        <ogc:Filter>
            <ogc:GmlObjectId gml:id="InWaterA_1M.1234"/>
        </ogc:Filter>
        </wfs:Query>
</wfs:GetFeature>
```

Abb. 9: Beispiel eines WFS GetFeature Requests (vgl. OGC, 2005)

Wie man an dem Beispiel erkennt, besitzt das Element GetFeature ein Query Element, in dem die Attribute des Featuretypen abgefragt bzw. Einschränkungen gemacht werden können. GetFeature Elemente können aber auch mehrere Query Elemente enthalten. Die Ergebnisse aller Queries werden verkettet und in ein result set geschrieben.

Um nähere Informationen zu den Featuretypen zu erhalten, kann die DescribeFeatureType Operation angewendet werden.

WFS Dienste sind vielseitiger einsetzbar als WMS Dienste. Beispielsweise können Clients nach Objekten suchen, die Suchergebnisse markieren oder die Daten, die ein Transaction WFS (WFS-T) liefert, manipulieren (vgl. MITCHELL, 2008).

4 Aufbau einer WebGIS Infrastruktur

4.1 MapServer UMN

Die Erstellung und Auslieferung von Karten im WWW übernehmen Kartendienste, die in der Literatur oft auch als Mapserver bezeichnet werden. Im folgenden Abschnitt wird MapServer UMN detailliert beschrieben.

MapServer UMN wurde seit Mitte der 1990er Jahre an der Universität von Minnesota (UMN) am Department of Forest Resources im ForNet Kooperationsprojekt mit der NASA und dem Minnesota Department of Natural Resources (MNDNR) entwickelt. In den Anfangsjahren der Entwicklung von MapServer UMN im ForNet Projekt wurden zwei unterschiedliche Kartendienste entworfen. Dies war zum einen der im NASA Umfeld mit der *ERDAS Imagine* C API programmierte *imgserv*, dessen Hauptaugenmerk auf der Visualisierung von Rasterdaten lag. Der über die CGI-Schnittstelle erreichbare Prozess benötigte als Eingabeparameter die räumliche Ausdehnung und die Bandkombination, um aus Landsat TM Daten Bilder im JPEG-Format zu erzeugen. Schließlich wurde dieses Programm mit der Funktionalität erweitert, die 24-bit Rasterdaten mit zusätzlichen 8-bit GIS-Daten zu überlagern. Daraus entstand die *mapserv* Anwendung (vgl. LIME, 2008).

Später wurde MapServer UMN vom TerraSip-Projekt übernommen, das von der NASA finanziert wurde. Mit dem Release von MapServer UMN 3.0 – gegen Ende des Projekts – wurde im Jahr 2000 der Quellcode offen gelegt. Seit kurzem wird MapServer UMN von der OSGeo betreut (vgl. http://mapserver.org).

Im Gegensatz zu vielen anderen Softwareprojekten, durchlief MapServer UMN nicht den formellen Entwicklungsprozess. Funktionen wurden, falls benötigt, der Einfachheit halber hinzugefügt oder durch das Einbinden externer Programmbibliotheken erweitert (vgl. LIME, 2008). Der Umstand, das Rad nicht neu erfinden zu müssen und sich aus Open Source Projekten bedienen zu können, trägt vielleicht zur Popularität von MapServer UMN bei. Zweifelsohne liegt der hohe Verbreitungsgrad darin begründet, dass keine Programmierkenntnisse erforderlich sind, um MapServer UMN zu konfigurieren.

4.1.1 Aufbau

MapServer UMN arbeitet mit einer Konfigurationsdatei (auch *Mapfile* genannt), CGI-Umgebungsvariablen und Vorlagen (templates). Im einfachsten Fall werden bei einer

Anfrage Parameter vom Webserver über die CGI-Schnittstelle an die MapServer UMN Anwendung mapserv übergeben, die daraufhin die Konfiguration lädt und je nach Modus eine Karte oder abgefragte Attribute im festgelegten HTML-Template zurückliefert. Folgende Abbildung veranschaulicht diesen Prozess:

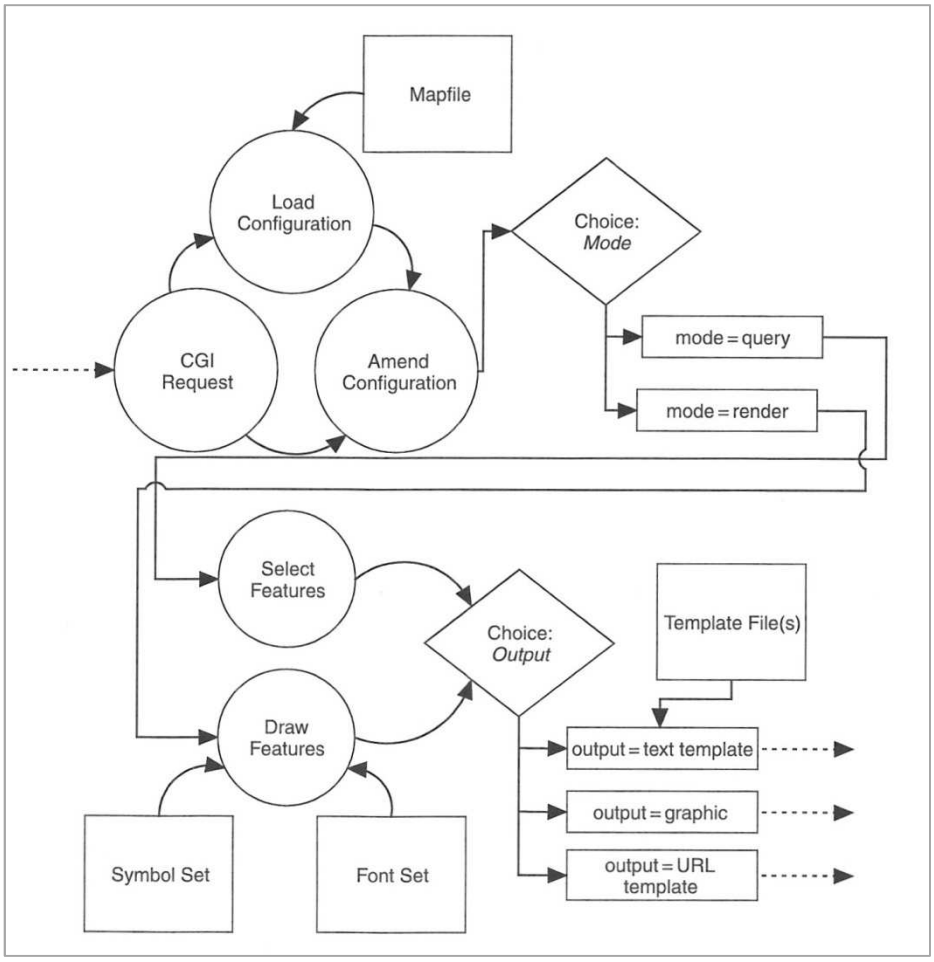


Abb. 10: MapServer UMN CGI Prozess (LIME, 2008)

Die Tags im Template werden in eckigen Klammern angegeben (z.B. [scalebar] für die Maßstabsleiste) und stellen die CGI-Variablen dar. MapServer ersetzt diese mit den erzeugten Elementen und liefert sie als Antwort zurück.

Anleitungen wie solch einfache HTML Clients zu entwerfen sind, finden sich in MITCHELL (2008) oder im MapServer Tutorial von NACIONALES und MCKENNA (2010) unter http://mapserver.org/tutorial/index.html.

4.1.2 Mapfile

Im Mittelpunkt einer MapServer UMN Anwendung steht das Mapfile, das wie bereits erwähnt, leicht mit einem Texteditor zu erstellen ist. In dieser Datei sind die essentiellen

Einstellungen enthalten. Dort werden Objektbeziehungen, die Datenquellen und die Ausgabeeinstellungen der zu erzeugenden Karte verwaltet.

Das Mapfile ist hierarchisch strukturiert und besteht aus Tags bzw. Schlüsselwörten. Diese definieren Objekte und müssen mit einem END Schlüsselwort enden. Das Mapfile ist case-insensitiv, d.h. die Groß- bzw. Kleinschreibung spielt keine Rolle. Die Wurzel der hierarchischen Struktur bildet das MAP Element. Sind Dateipfade angegeben, so können diese relativ zum Speicherort des Mapfiles oder mit absolutem Pfad angegeben werden. Kommentare müssen mit einem # gekennzeichnet werden.

Das Mapfile beginnt mit einem Header, der die räumliche Ausdehnung der Karte definiert und weitere elementare Objekte, wie Maßstabsleiste, Projektion oder Ausgabeformat, enthalten kann. Einige Tags können auch mehrfach im Mapfile vorkommen und beziehen sich dann auf das Objekt in dessen Struktur sie eingebunden sind. So kann beispielsweise das Schlüsselwort PROJECTION sowohl im MAP Objekt als auch in einzelnen LAYER Objekten angeführt sein.

Nachfolgend ist ein kurzes Beispiel eines Mapfiles angeführt, das mit dem MapServer Export Tool aus Quantum GIS, einem Open Source Desktop GIS Client (http://qgis.org/), erzeugt wurde. Dargestellt wird ein Polygon Layer, der rot symbolisiert wird. Die Karte besitzt die Projektion Gauss Krüger Zone 4 mit dem EPSG-Code 31468 und wird im JPEG-Format ausgeliefert.

```
MAP
  NAME "QGIS-MAP"
  # Map image size
  SIZE 1024 512
  UNITS meters
  EXTENT 4445475.181419 5396170.224775 4472152.458028
   5409675.695363
  FONTSET 'c:/mapfile/fonts/fonts.txt'
  SYMBOLSET 'c:/mapfile/symbols/symbols.txt'
  PROJECTION
    'proj=tmerc'
    'lat_0=0'
    'lon_0=12'
    'k=1'
    'x_0=4500000'
    'y_0=0'
    'ellps=bessel'
    'datum=potsdam'
    'units=m'
    'no_defs'
  END
#...
  OUTPUTFORMAT
    NAME jpeg
    DRIVER 'GD/JPEG'
    MIMETYPE 'image/jpeg'
    IMAGEMODE RGBA
    EXTENSION 'jpeg'
  END
#...
LAYER
   NAME “Testlayer”
   TYPE POLYGON
   DATA “c:/temp/testshape.shp”
   STATUS ON
   CLASS
      STYLE
      COLOR 255 0 0
      END  #STYLE
   END # CLASS
END #LAYER
END #MAP
```

Dieses kleine Beispiel soll die Hierarchie der einzelnen Objekte in einem Mapfile verdeutlichen. Zur besseren Lesbarkeit wurden die zu einzelnen Objekten gehörenden Attribute eingerückt. Zeilenvorschübe sind nicht notwendig. Aufgrund dessen können Objekte mit ihren Attributen in einer Zeile definiert werden.

Detaillierte Ausführungen zu den einzelnen Objekten sind in der MapServer UMN Mapfile Referenz zu finden (http://mapserver.org/mapfile/index.html). Es wird an dieser Stelle ausdrücklich darauf hingewiesen, dass in der vorliegenden Arbeit nur diejenigen Objekte und Konfigurationen von MapServer UMN angeführt sind, die für die Fragestellung von Bedeutung sind.

4.1.2.1 Layerdefinitionen

Das Layerkonzept von MapServer UMN unterscheidet sich nicht zu dem allgemein in der Geoinformatik verwendeten. Ein Layerobjekt enthält die Informationen, die MapServer UMN benötigt, um auf die Datenquellen zuzugreifen, sie abfragen und darstellen zu können (vgl. LIME, 2008).

Layer werden im Mapfile mit dem LAYER Schlüsselwort angegeben. Der Layertyp wird mit dem Tag TYPE angegeben und kann folgende Ausprägung besitzen:

Typ	Beschreibung
point	durch Symbole dargestellte Punkte
line	Darstellung von Linienlayern
polygon	Darstellung von geschlossenen Polygonen; um Polylinien zu erzeugen, muss das COLOR Schlüsselwort weggelassen werden
circle	Kreislayer: Ein Kreis wird definiert durch die Angabe seines minimal einschließenden Rechtecks
annotation	Beschriftungslayer: Die Geometrie des Objekts wird nicht dargestellt. Stattdessen wird ein Einfügepunkt berechnet, der beschriftet wird. Optional kann dieser Punkt dargestellt werden. Es können Punkt-, Polygon- und Linienlayer beschriftet werden.
raster	Rasterlayer: Rasterdaten
query	Abfragelayer: Diese Layer werden nicht visualisiert. Sie dienen nur zur Abfrage.
chart	Diagrammlayer: Seit MapServer UMN 5.0 eingeführter Layertyp, der nur mit der AGG oder der GD Bibliothek funktioniert und sowohl Balken- als auch Tortendiagramme unterstützt.

Tab. 6: Layertypen im Mapfile (in Anlehnung an KORDUAN und ZEHNER, 2008)

Neben dem Typ muss selbstverständlich auch die Datenquelle des Layers angegeben werden. Es können folgende Formate eingebunden werden (vgl. http://mapserver.org/input/index.html):

- Vektordaten
  - ESRI Shapefiles
  - PostGIS/PostgreSQL
  - von OGR unterstützte Vektordaten (siehe Anlage 2)
  - MapInfo
  - WFS
  - GML
  - Virtual Spatial Data
  - ArcInfo
  - ArcSDE
  - DGN
  - S57
  - ESRI Personal Geodatabase
  - Inline
  - KML
  - Oracle Spatial
  - MySQL
  - MSSQL 2008
  - NTF
  - SDTS
  - USGS TIGER
  - GPS Exchange Format
- Rasterdaten
  - TIFF/GeoTIFF
  - GIF
  - PNG
  - JPEG
  - Erdas .LAN/.GIS
  - von GDAL unterstützte Formate (siehe Anlage 1)

Da für die Unterstützung dieser Eingabeformate MapServer UMN unterschiedlich kompiliert werden muss und deshalb auch bereits vorkompilierte Installationspakete wie MS4W (http://www.maptools.org/ms4w/index.phtml) oder OSGeo4W (http://trac.osgeo.org/osgeo4w/) zum Teil verschiedene Formate unterstützen, empfiehlt es sich, ins Anwendungsverzeichnis von MapServer UMN zu wechseln und die Anwendung mit dem Laufzeitparameter –v zu starten und sich die unterstützten Formate auflisten zu lassen:

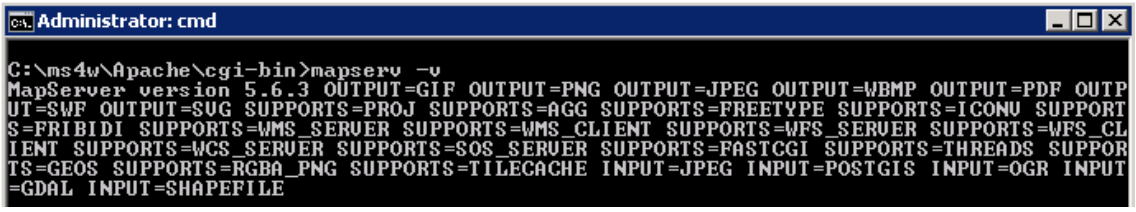


Abb. 11: Formatunterstützung MapServer UMN (eigener Entwurf)

Für die unterschiedlichen Datenquellen, muss im Mapfile der jeweils benötigte Typ des Schlüsselworts CONNECTIONTYPE angegeben werden. Dieses kann die Ausprägungen "local", "sde", "ogr", "postgis", "oraclespatial", "wms", "wfs" oder "plugin" besitzen. Bei datenbankbasierten Quellen muss zusätzlich das CONNECTION Tag verwendet werden, das die Informationen über den Zugriff zur Datenbank enthält. Der Dateiname bzw. Tabellenname wird im DATA Tag angegeben. An dieser Stelle sei auf die entsprechenden Stellen in der MapServer UMN Dokumentation verwiesen. Ein einfaches Beispiel wurde bereits unter 4.1.2 aufgezeigt.

4.1.2.2 Klassifizierungen und Styles

Jeder zu visualisierende Layer benötigt im Mapfile ein entsprechendes CLASS Schlüsselwort. Damit lassen sich Attribute eines Layers zu Klassen zusammenfassen. Es können sowohl Vektor- als auch Rasterdaten klassifiziert werden. Anhand von Ausdrücken besteht die Möglichkeit, Objekte zu gruppieren. MapServer UMN verwendet dafür das Tag EXPRESSION: Drei unterschiedliche Varianten von Ausdrücken werden unterstützt. Die einfachste Form stellen Textvergleiche über ein Attributfeld dar, z.B.

```
CLASSITEM "LANDNUTZUNG"
    CLASS
      NAME "Acker"
      EXPRESSION 'Acker'
       STYLE
         WIDTH 0.91
         OUTLINECOLOR 0 0 0
         COLOR 170 85 0
       END
    END
```

Das Attribut, das die zu klassifizierenden Werte enthält, wird mit CLASSITEM angegeben. Des Weiteren werden reguläre Ausdrücke und logische Ausdrücke unterstützt. Wie im Klassifizierungsbeispiel zu sehen ist, sind die stilistischen Visualisierungsinformationen, wie Farbe, Symbolart und -größe dem STYLE Tag zugeordnet. Ein CLASS Objekt kann mehrere Stilangaben besitzen. Damit lassen sich kartographisch

ansprechende Darstellungen realisieren. Beispielsweise kann damit ein zusammengesetztes Linienobjekt erzeugt werden. MapServer UMN rendert die Stile der Reihenfolge nach – von oben beginnend (vgl. LIME, 2008). Mit folgendem Codebeispiel wird ein Linienobjekt zuerst schwarz mit Größe 5 dargestellt. Darüber wird eine rote Linie der Größe 4 gerendert, gefolgt von einer dünnen schwarzen ().

```
CLASS
  NAME 'Nebenstrasse'
    STYLE
   SYMBOL "solid"
    SIZE 5
    COLOR 0 0 0
  END #STYLE
  STYLE
   SYMBOL "solid"
    SIZE 4
    OUTLINECOLOR 255 0 0
  END #STYLE
  STYLE
   SYMBOL "solid"
    SIZE 1
    COLOR 0 0 0
  END #STYLE
 END #CLASS
```

Beschriftungen werden mit LABEL Objekten umgesetzt. Das LABELITEM Schlüsselwort gibt dabei das Beschriftungsfeld an.

Um einen Überblick über die Mapfile Konfiguration und die darin enthaltenen Hierarchien zu erhalten, ist in der folgenden Abbildung das MapServer UMN Objektmodell dargestellt.

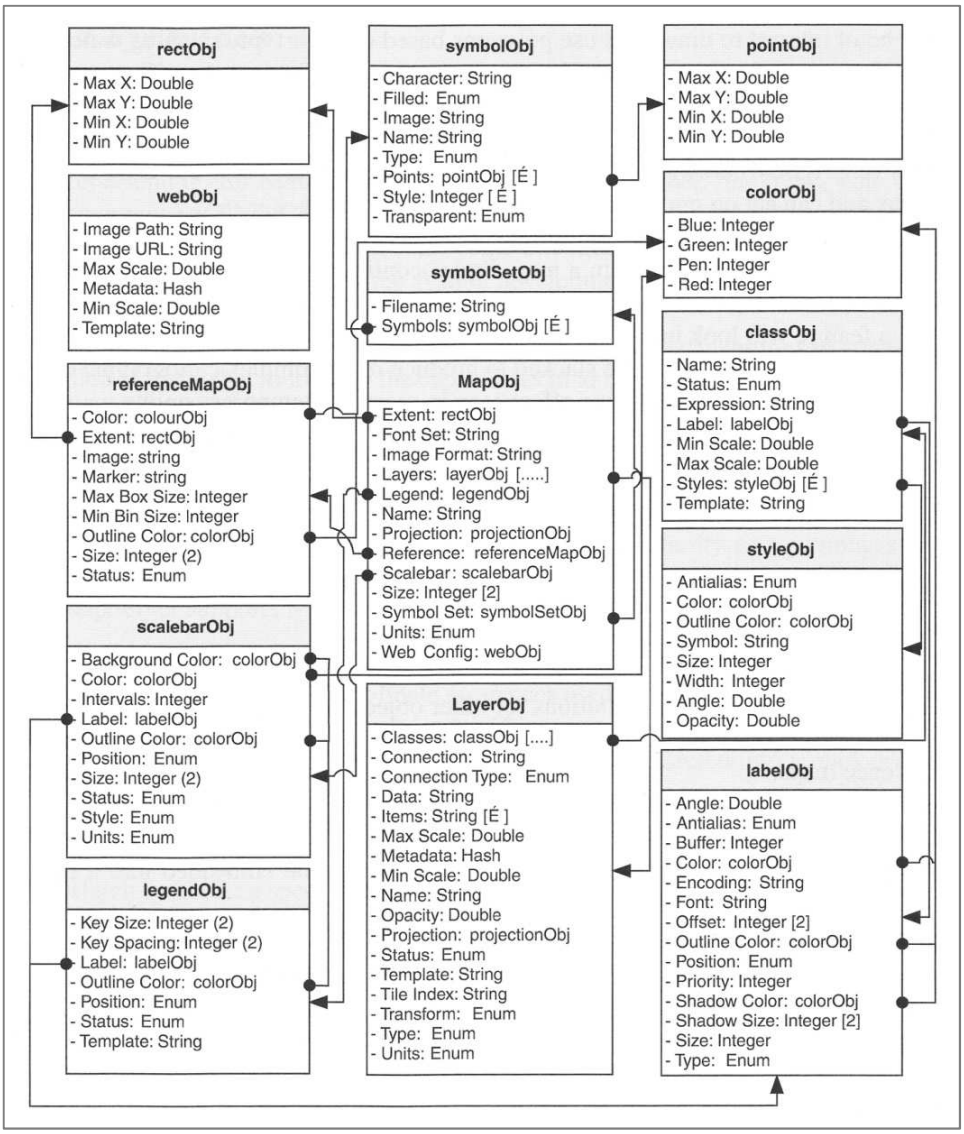


Abb. 12: MapServer UMN Objektmodell (LIME, 2008)

4.1.2.3 Ausgabeformat

MapServer UMN bietet die Möglichkeit, Bilder in unterschiedlichen Formaten auszugeben. Dies können zum einen gängige Rasterdatenformate, wie JPEG, GIF oder PNG zum anderen Formate, wie PDF, SWF und SVG sein. Diese werden im Mapfile mit dem Tag OUTPUTFORMAT angegeben. Die Ausgabetypen sind dabei abhängig von den verwendeten Rendering Bibliotheken. Der zu verwendende Treiber ist jeweils mit dem DRIVER Schlüsselwort und dem eigentlichen Format anzugeben. Die wichtigsten Bibliotheken sind *GD*, *AGG* und *GDAL*.

Erstere wird auch als *libgd* bezeichnet und das Kürzel GD sollte ursprünglich „gif draw“ bedeuten. Diese Abkürzung wurde allerdings mit der Patentierung der Kompressionsmethode von GIF Bildern verworfen. Seither kann von „graphics draw“ gesprochen werden. GD unterstützt das JPEG-, PNG-, WBMP- und GIF-Format (vgl. http://www.libgd.org). Im Mapfile werden diese als GD/JPEG, GD/PNG, GD/WBMP oder GD/GIF angegeben (vgl. LIME et al., 2011).

AGG, das für Anti-Grain Geometry steht, wird von MapServer UMN seit der Version 5.0 unterstützt und kann JPEG und PNG Bilder erzeugen. Vor allem für das Anti-Aliasing wird AGG in der MapServer UMN Dokumentation explizit empfohlen (vgl. NACIONALES, 2009). Auch hier wird der Treiber dem Format vorangestellt, z.B. AGG/JPEG. Details zu AGG sind unter http://www.antigrain.com/ zu finden.

Da GDAL nicht nur für das Ausgabeformat von Bedeutung ist, wird die Bibliothek unter Abschnitt 4.1.3 erläutert.

Neben dem Bildformat können auch unterschiedliche Bildmodi mit dem Tag IMAGEMODE angegeben werden. Möglich sind folgende – je nach Treiberunterstützung:

Typ	Beschreibung
PC256	256 Farben
RGB	24bit rot/grün/blau ohne Transparenzunterstützung
RGBA	32bit rot/grün/blau/alpha mit Transparenzunterstützung
BYTE	8bit Bilder, nur für Layertyp Raster
INT16	16bit Bilder, nur für Layertyp Raster
FLOAT32	32bit Bilder, nur für Layertyp Raster

Tab. 7: IMAGEMODE Varianten (vgl. LIME et al., 2011)

4.1.3 GDAL / OGR

Wie im letzten Abschnitt bereits kurz angerissen, spielen externe Bibliotheken für MapServer UMN eine entscheidende Rolle. Sie erweitern den Kartendienst um eine Vielzahl essentieller Funktionen. Zu nennen sind in diesem Zusammenhang *Proj.4* für Projektionen, *Geometry Engine Open Source (GEOS)* für räumliche Funktionen oder *libcurl*, die die Grundlage für die OGC WMS/WFS/WCS Client und Server Unterstützung bildet. Eine Übersicht über weitere externe Bibliotheken gibt KROPLA (2005). Für

die Interoperabilität von MapServer UMN ist vor allem eine Bibliothek unverzichtbar – die *Geospatial Data Abstraction Library (GDAL).*

GDAL ist eine in C++ geschriebene Bibliothek, um lesenden und schreibenden Zugriff auf eine Vielzahl verschiedener Datenformate zu erhalten. Die Bibliothek ist zweigeteilt. Zum einen gibt es für Rasterdatenzugriffe GDAL, zum anderen die für den Vektordatenzugriff zuständige Komponente *OGR Simple Features Library (OGR)*. Beide besitzen ein eigenes Objektmodell und eine eigene Programmierschnittstelle. GDAL ist Open Source und ist deshalb in vielen Open Source Software Paketen, wie z.B. Quantum GIS, GRASS GIS oder OpenEV integriert. Aber auch proprietäre Produkte wie die mächtige Konvertierungssoftware Feature Manipulation Engine von Safe Software oder ArcGIS von ESRI greifen auf GDAL zurück (vgl. WARMERDAM, 2008).

GDAL in der Version 1.7.0 unterstützt 99 verschiedene Rasterdatenformate und OGR 34 verschiedene Vektordatenformate (siehe Anlagen 1 und 2). Des Weiteren liefert GDAL eine Reihe nützlicher Werkzeuge für die Kommandozeile. Nachfolgende Tabelle listet ausgewählte Tools auf, die für die Rasterdatenverarbeitung verwendet werden können:

Anwendung	**Beschreibung**
gdalinfo	gibt nützliche Informationen bzw. Metadaten zu einer Rasterdatei aus, wie z.B. verwendeter Treiber, Größe, Koordinatensystem
gdal_translate	konvertiert Rasterdaten in ein anderes Format und unterstützt auch Resampling oder Größenänderungen
gdaladdo	erstellt Bildpyramiden
gdalwarp	projiziert Raster in ein anderes Koordinatensystem
gdaltindex	erstellt einen MapServer UMN Tileindex im ESRI Shape Format
gdalbuildvrt	erzeugt ein Virtual Raster Table
rgb2pct.py	Python Skript, um 24bit RGB Rasterdaten in 8bit umzuwandeln
pct2rgb.py	Python Skript, um 8bit Rasterdaten in 24bit RGB umzuwandeln
gdal_merge.py	Python Skript, das mehrere Eingangsraster zu einem Zielraster zusammenfügt
gdal2tiles.py	Python Skript, um aus einem Raster Kacheln nach der OSGeo Tile Map Service Spezifikation (TMS) zu erzeugen
gdaltransform	transformiert Koordinaten in ein anderes Bezugssystem

Tab. 8: ausgewählte GDAL Tools (vgl. http://www.gdal.org/gdal_utilities.html)

Die Beschreibung weiterer Tools ist auf der GDAL Homepage http://www.gdal.org/gdal_utilities.html zu finden. OGR beinhaltet drei Werkzeuge:

Anwendung	Beschreibung
ogrinfo	gibt nützliche Informationen bzw. Metadaten zu einer Vektordatei aus
ogr2ogr	konvertiert Vektordaten in ein anderes von OGR unterstütztes Format
ogrtindex	erstellt einen Tileindex zur Verwendung in MapServer UMN

Tab. 9: OGR Tools (vgl. http://www.gdal.org/ogr_utilities.html)

4.1.4 Nutzung von OGC Webservices mit MapServer UMN

MapServer UMN unterstützt eine Reihe von OGC Web Services, die für interoperable Systemlandschaften von großer Bedeutung sind. MapServer UMN in der Version 5.6.6 unterstützt folgende OGC Spezifikationen (verändert nach http://mapserver.org/faq.html):

- Web Map Service (Client / Server)
- Web Feature Service (Client / Server)
- Web Coverage Service (Server)
- Geography Markup Language
- Web Map Context Documents
- Styled Layer Descriptor
- Filter Encoding Specification
- Sensor Observation Service (Server)
- Observations and Measurements
- SWE Common
- OWS Common

Für die vorliegende Arbeit ist MapServer UMN als WMS Client und Server von großer Bedeutung. Aus diesem Grund werden im folgenden Abschnitt die Konfigurationsschritte aufgezeigt.

MapServer UMN unterstützt WMS in den Versionen 1.0.0, 1.0.7, 1.1.0, 1.1.1 und 1.3.0. Um MapServer UMN als WMS Client oder auch Server einzusetzen, sind lediglich Einträge im Mapfile hinzuzufügen, vorausgesetzt MapServer UMN wurde mit WMS Unterstützung kompiliert.

Folgende Tabelle enthält alle zwingend benötigten LAYER und METADATA Parameter, um einen WMS Layer ins Mapfile zu integrieren:

Parameter	Beschreibung
CONNECTIONTYPE WMS	
CONNECTION	Basis URL des WMS Servers ohne GET Parameter
"wms_format"	MIME-Type des angeforderten Bildes, z.B. image/png
"wms_name"	kommagetrennte Liste der anzufordernden WMS Layer; sowohl die GET Parameter LAYERS als auch QUERY_LAYERS werden verwendet
"wms_server_version"	Version des WMS Servers
"wms_srs"	leerzeichengetrennte Liste mit EPSG Codes, die vom WMS Server unterstützt werden

Abb. 13: WMS Layer Integration im Mapfile (vgl. MCKENNA, 2010a)

Eine Beispielkonfiguration für einen WMS Layer im Mapfile könnte wie folgt aussehen (vgl. MCKENNA, 2010a):

```
LAYER
  NAME "continents"
  TYPE RASTER
  STATUS ON
  CONNECTION "http://demo.mapserver.org/cgi-bin/wms?"
  CONNECTIONTYPE WMS
  METADATA
    "wms_srs"             "EPSG:4326"
    "wms_name"            "continents"
    "wms_server_version"  "1.1.1"
    "wms_format"          "image/jpeg"
  END
END
```

Um MapServer UMN als WMS Server einzurichten, müssen neben Anpassungen im LAYER Objekt auch WMS-spezifische Einstellungen im MAP Objekt des Mapfiles gemacht werden. Dabei reicht ein Blick auf die Anforderungen an ein WMS GetCapabilities, um zu wissen, welche Parameter angegeben werden müssen. Es gilt zu beachten, dass jede Instanz des WMS Servers ein eigenes Mapfile benötigt.

Nachfolgende Tabelle fasst alle vorgeschriebenen Parameter und Metadaten zusammen:

Objekt	**Parameter**
MAP - **WEB**	NAME
MAP - **WEB**	PROJECTION
MAP - **WEB**	METADATA - wms_title - wms_onlineresource - wms_srs
LAYER	NAME
LAYER	PROJECTION
LAYER	METADATA - wms_title - wms_srs
LAYER	STATUS
LAYER	TEMPLATE
LAYER	DUMP TRUE

Tab. 10: WMS Server Integration im Mapfile (vgl. McKenna, 2010b)

Bei Angabe des Raumbezugssystems im MAP Objekt ist zu beachten, dass falls der EPSG Projektionscode in der Form "init=epsg:xxxx" verwendet wird, MapServer UMN ein BoundingBox Tag im WMS Capabilities Dokument erzeugt. Dann ist der Parameter "wms_srs" im METADATA Tag des MAP Objekts überflüssig. Es ist jedoch trotzdem sinnvoll diesen Parameter anzugeben, vor allem dann, wenn die Quelldaten seltene Projektionen besitzen. In diesem Fall sollte bei PROJECTION der korrekte EPSG Code und bei "wms_srs" gängige Raumbezugssysteme angegeben werden. Die im MAP Objekt angegebene Projektion wird auf die einzelnen Layer übertragen. Es empfiehlt sich jedoch auch diese Parameter mit anzugeben. Für PROJECTION und "wms_srs" Parameter im LAYER Objekt gelten die gleichen Vorgaben wie im MAP Objekt. Der Parameter "wms_onlineresource" wird ebenso für das Capabilities Dokument benötigt. Wird

er nicht angegeben, so generiert MapServer UMN eine Standard URL aus dem Host- und Skriptnamen.

Folgendes Codebeispiel zeigt die nötige Konfiguration im WEB Objekt des Mapfiles (McKenna, 2010b):

```
NAME "DEMO"
...

WEB
  ...
  METADATA
    "wms_title"           "WMS Demo Server"
    "wms_onlineresource"  "http://my.host.com/cgi-
bin/mapserv?map=wms.map&"
    "wms_srs"             "EPSG:4269 EPSG:4326"
  END
END

PROJECTION
  "init=epsg:42304"
END
...
END
```

Ein LAYER TEMPLATE muss für GetFeatureInfo Requests an den WMS Server angegeben werden. Wird in diesem Request GML angefordert, so muss der Layer mit DUMP TRUE angegeben werden (vgl. McKenna, 2010b). Für alle optionalen Parameter wird auf die MapServer UMN Dokumentation verwiesen.

4.2 Clients

Nachdem die grundlegenden Technologien beschrieben wurden und MapServer UMN als Kartenserver vorgestellt wurde, wird im kommenden Abschnitt ein Blick auf die Clientseite geworfen. Man kann browserbasierte und desktopbasierte WebGIS Clients unterscheiden. Unter letzteren versteht man Desktopanwendungen, die OGC konforme Webdienste einbinden können, z.B. Quantum GIS oder gvSIG. Auf diese wird in der vorliegenden Arbeit nicht näher eingegangen.

Bei den browserbasierten WebGIS Clients zeigt sich seit Google Maps und Co. ein Trend hin zum Einsatz von AJAX. Unter dem Begriff *Asynchronous JavaScript and XML* sind mehrere Technologien vereint. Dazu gehören die Verwendung von Extensible Hypertext Markup Language (XHTML), Cascading Style Sheets (CSS), Document Object Model (DOM), asynchroner Datenempfang per XMLHttpRequest und JavaScript Binding (vgl. Garrett, 2005). Bei herkömmlichen Webapplikationen folgt auf

eine Aktivität des Clients eine Antwort des Servers. Diese Wechselwirkung wird als synchron bezeichnet. Bei Ajax ist noch eine in JavaScript geschriebene Zwischenschicht, die Ajax Engine, integriert. Diese ist im Hintergrund tätig und für die Clientoberfläche und die Kommunikation mit dem Server verantwortlich. Sie bewirkt asynchrone Interaktionen zwischen Client und Server (vgl. GARRETT, 2005). Dies bedeutet, dass nicht bei jeder Interaktion die Seite neu geladen werden muss. Im Falle von WebGIS Clients bedeutet das beispielsweise, dass ein Anwender eine Karte mit Kommentaren versehen kann, ohne dass jedes Mal die Karte vom Server neu angefordert werden müsste. Folgende Abbildung stellt den klassischen Ansatz (oben) dem von Ajax (unten) gegenüber:

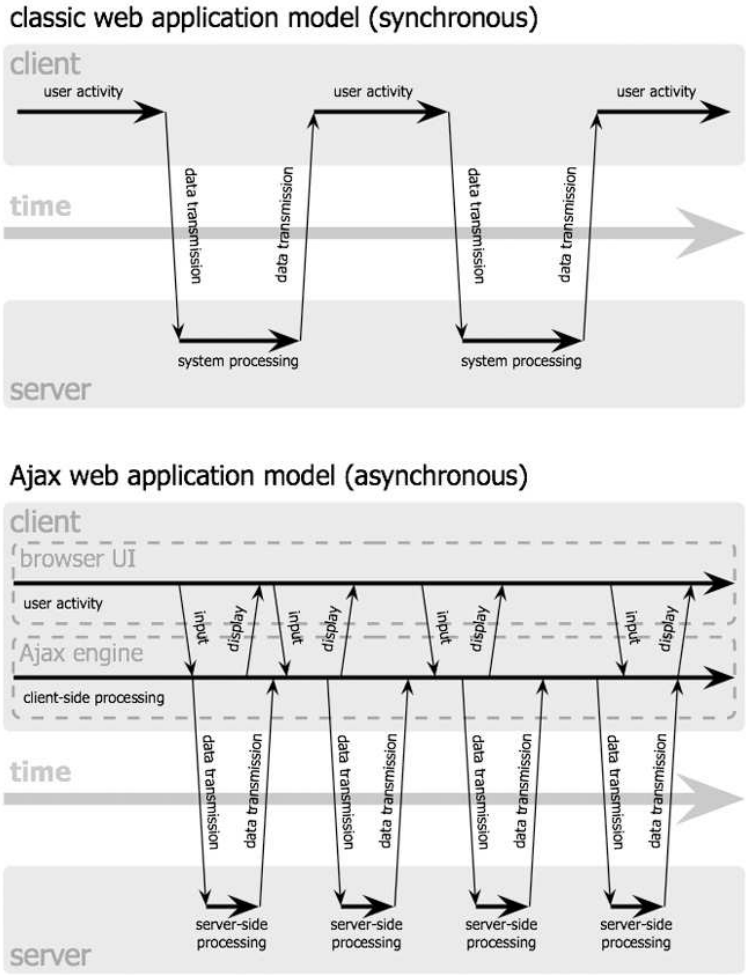


Abb. 14: Vergleich klassisches Webapplikationsmodell zu Ajax (GARRETT, 2005)

Zwei Clients, die Ajax verwenden, werden im Folgenden vorgestellt. Zum einen wird kurz auf das Open Source Produkt OpenLayers eingegangen, zum anderen wird mapAccel von Territorium Online GmbH vorgestellt, mit dem letztendlich die Umsetzung erfolgen soll.

4.2.1 OpenLayers

Bei OpenLayers handelt es sich um eine im Jahr 2006 von MetaCarta veröffentlichte JavaScript Bibliothek, die es ermöglicht Kartendienste und eigene Geodaten browserbasiert zu visualisieren. OpenLayers stellt dabei Tools bzw. Bedienelemente bereit, mit denen die Karte erkundet werden kann. Die JavaScript Bibliothek bietet eine API mit der Kartendienste sehr einfach integriert werden können. Diese nutzt zum Teil auch andere Bibliotheken, wie *Prototype, Rico, XMLHttpRequest.js* und *Gears*. OpenLayers verfolgt dabei das Konzept, die Daten und die Interaktionen zu trennen. Dies ermöglicht den flexiblen Austausch von Kartendiensten ohne Neuimplementierung von Interaktionen (vgl. JANSEN und ADAMS, 2010).

Um OpenLayers nutzen zu können, reicht es aus, die OpenLayers Bibliothek im <head> Tag einer HTML Seite einzubinden. Dabei ist auch Hotlinking, der direkte Verweis auf die Online Ressource, erlaubt. Die Bibliothek kann aber auch von einem lokalen Server eingebunden werden (vgl. JANSEN und ADAMS, 2010).

Um die Bibliothek einzubinden, muss folgender Code in eine HTML Seite integriert werden:

```
<script
src="http://www.openlayers.org/api/2.10/OpenLayers.js"
type="text/javascript"></script>
```

Im Beispiel wird die Version 2.10 verwendet. Dies ist entsprechend der gewünschten Version anzupassen. Des Weiteren muss in der HTML Seite ein Element enthalten sein. Dieses sollte auch ein Attribut defer="defer" besitzen, damit der Code im Script Element erst dann interpretiert wird, wenn das DOM bereit ist. Um eine einfache Applikation mit nur einem Layer zu erzeugen, muss ein OpenLayers Kartenobjekt und ein OpenLayers Layerobjekt erzeugt werden. Das Layerobjekt muss anschließend mit der Methode *addLayer* der Karte hinzugefügt werden (vgl. JANSEN und ADAMS, 2010).

Ein einfaches Beispiel zur Integration eines WMS Dienstes in OpenLayers zeigt folgende Abbildung (verändert nach JANSEN und ADAMS, 2010):

```
<html>
   <head>
   <title>OpenLayers WMS Beispiel</title>
       <script src="http://www.openlayers.org/api/2.10/OpenLayers.js" type="text/javascript"></script>
   </head>
   <body>
       <div id ="map" style="width:100%; height:100%; border: 3px solid #888">
       </div>
       <script defer="defer" type="text/javascript">
           var map = new OpenLayers.Map('map');
           var wms = new OpenLayers.Layer.WMS("Demo WMS", "http://demo.mapserver.org/cgi-bin/wms?",
                   {layers: 'continents'});
           map.addLayer(wms);
           map.zoomToMaxExtent();
       </script>
   </body>
</html>
```

Abb. 15: OpenLayers Codebeispiel (eigener Entwurf)

Mit der Methode zoomToMaxExtent wird auf den maximalen Kartenausschnitt gezoomt. Der OpenLayers Client aus dem Beispielcode sieht schließlich wie folgt aus:

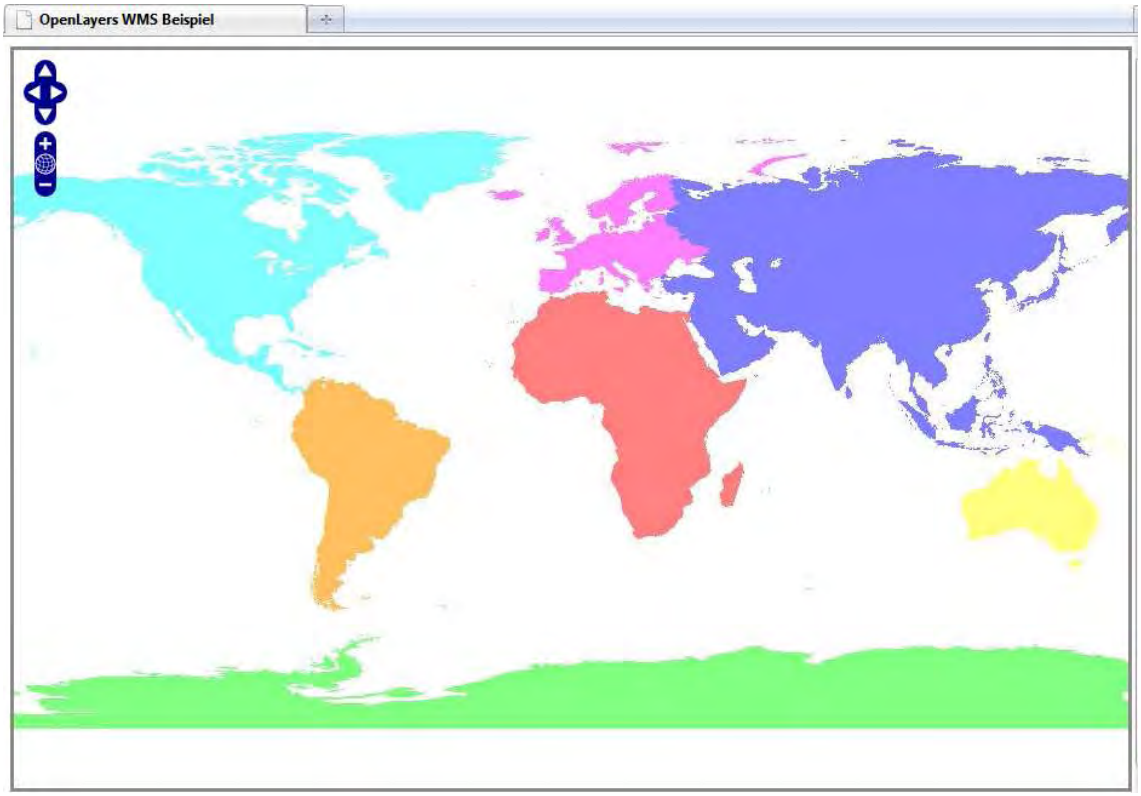


Abb. 16: OpenLayers Client (eigener Entwurf)

Für detaillierte Beschreibungen der OpenLayers Objekte und Methoden wird auf JANSEN und ADAMS (2010) verwiesen.

Eine zweite Möglichkeit OpenLayers in eine Webseite einzubinden, ist die Verwendung des OpenLayers Map Viewer Service. In diesem können points of interest, die zuvor in einer Textdatei mit Koordinatenangabe und einer Beschreibung versehen wurden, vi-

sualisiert werden. Aufgrund der Einfachheit und der fehlenden Möglichkeit andere Dienste zu integrieren, wird darauf nicht näher eingegangen. Ein Codebeispiel findet sich unter KORDUAN und ZEHNER (2008). Ausführlicher wird der Map Viewer Service unter http://trac.osgeo.org/openlayers/wiki/MapViewerService beschrieben.

OpenLayers stellt ein flexibles Werkzeug dar, um ohne großen Aufwand Webservices einbinden zu können. Unter http://openlayers.org/dev/examples/ finden sich Beispiele zu folgenden OGC Spezifikationen: WMS, WFS, WMTS, Sensor Observation Service (SOS), GML, Web Map Context Documents, SLD, Filter Encoding Specification.

4.2.2 mapAccel

Der mapAccel WebGIS Client ist Teil des komplexen mapAccel WebGIS Applikation Frameworks von Territorium Online. Dabei nimmt der Client, wie bereits anhand des Schichtenmodells unter 3.1 beschrieben, die Präsentationsschicht ein. Der browserbasierte Smart Mapping Client besitzt ein eigenes Datenmodell und besteht nur aus HTML, JavaScript und CSS. Die Entwicklung erfolgte mit dem Google Web Toolkit (GWT) Framework.

Der Client selbst besitzt folgenden Aufbau:

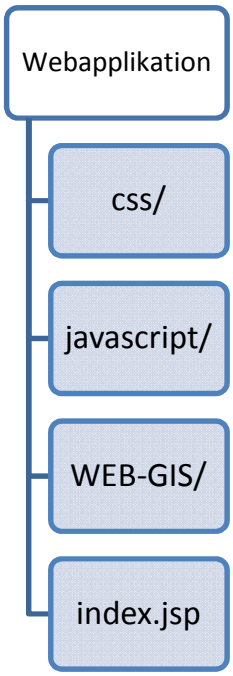


Abb. 17: Struktur des mapAccel WebGIS Clients (verändert nach TERRITORIUM ONLINE GMBH, 2010a)

Die Datei *index.jsp* enthält dabei die Verlinkungen auf die Stylesheets und JavaScript Dateien, besitzt aber auch HTML. Der Ordner WEB-GIS enthält alle Kernfunktionalitäten (JavaScript und CSS) (vgl. TERRITORIUM ONLINE GMBH, 2010a).

Der WebGIS Client spricht den mapAccel Applikationsserver in JSON an. Folgende Abbildung zeigt schematisch die mapAccel Architektur:

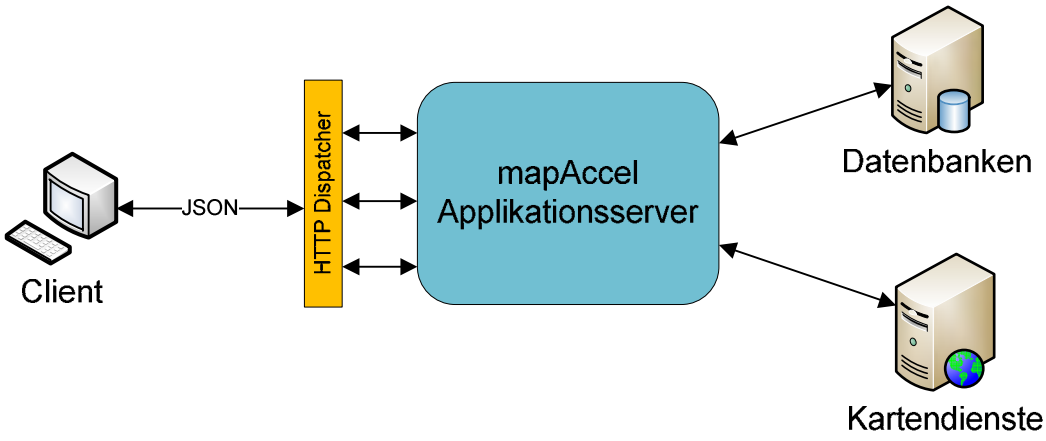


Abb. 18: Architektur von mapAccel (verändert nach TERRITORIUM ONLINE GMBH, 2010a)

Der Applikationsserver ist modular nach dem Open Standard Gateway Interface (OSGi) aufgebaut und läuft innerhalb eines OSGi Containers. Mit Hilfe des OSGi Frameworks lassen sich javabasierte Module bzw. bundles entwickeln und in einen größeren Applikationskontext stellen. Die Module sind dabei zusammenhängend, aber kaum aneinander gekoppelt. Dies hat den Vorteil, dass Module einfach ausgetauscht werden können. Die anderen Module werden dabei kaum oder überhaupt nicht beeinflusst (vgl. WALLS, 2009).

Der mapAccel Applikationsserver kann von jedem anderen Client über einen Representational State Transfer (RESTful) Service angesprochen werden. Web Services werden in dieser Architektur als Ressourcen betrachtet und können über ihre URL angesprochen werden (vgl. TYAGI, 2006). Um beispielsweise eine Karte abzufragen, kann folgende URL aufgerufen werden:

```
http://irgendeineip/irgendeinewebapp/webservice/map?Format
=json&bbox=4480000,5287000,4490000,5297000&size=512,512
```

Die Antwort des Service mit der URL des Kartenbildes lautet in JSON:

```
{"type":"Response",
"data":{
"type":"MapImage",
"bbox":[4480000,5287000,4490000,5297000],
"scale":73818.89763779528,
"url":"http://irgendeineip/output/12983926713696180.jpg",
"width":512,
"height":512
}
}
```


Abb. 19: JSON Antwort von mapAccel (eigener Entwurf)

Der mapAccel Applikationsserver kann sowohl mit dem kommerziellen Kartendienst ArcIMS von ESRI über ArcXML als auch mit mapAccel IMS über MapXML kommunizieren. Die Basis von mapAccel IMS bildet MapServer UMN. Aus diesem Grund können alle bereits genannten Vorteile im Hinblick auf Offenheit, Interoperabilität, Stabilität und Performance genutzt werden.

Um eine Konfiguration für den Applikationsserver zu erstellen, wird das mapAccel Backoffice *mapSnap* benötigt. Ausgehend von einer bestehenden Kartendienstkonfiguration, beispielsweise eines Mapfiles, können damit Zusatzinformationen hinzugefügt, Abfragen konfiguriert, das Aussehen und die Verhaltensweise des Clients verändert und Zugriffsrechte vergeben werden (vgl. TERRITORIUM ONLINE GMBH, 2010b).

Die Benutzeroberfläche des Clients ist übersichtlich gestaltet und besteht im Wesentlichen aus einer großen Karte, einer ausblendbaren Übersichtskarte, einer Werkzeugleiste, dem Inhaltsverzeichnis mit Legende und Export Tool sowie einem Werkzeug für verschiedene Zoomstufen.

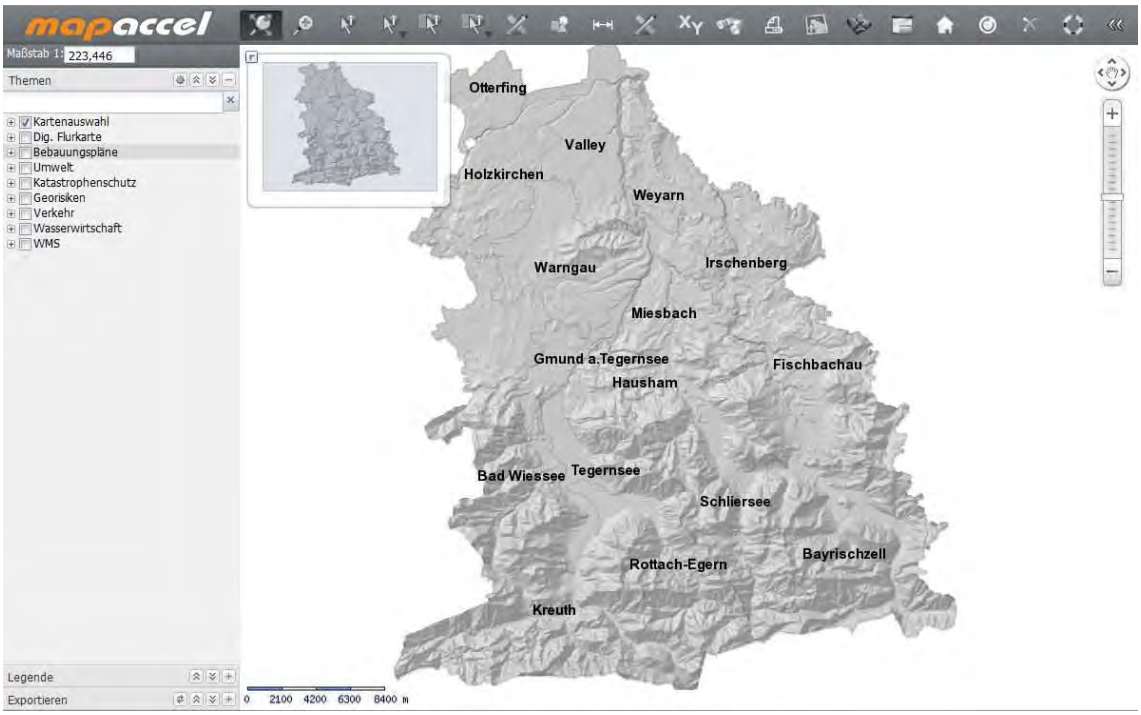


Abb. 20: mapAccel WebGIS Client (LANDRATSAMT MIESBACH, 2011)

4.3 Datenmanagement

Neben der Wahl des Kartendienstes und des Clients, spielt die Datenhaltung und das Datenmanagement eine entscheidende Rolle. Ohne eine genaue Prüfung, ob sich ein bestimmtes Datenmodell oder –format für die gestellten Anforderungen eignet, sollte nicht mit der Umsetzung begonnen werden (vgl. KORDUAN und ZEHNER, 2008).

Für die Haltung und das Management von Vektordaten haben sich objektrationale Datenbanksysteme mit Erweiterungen für die Verarbeitung von Geodaten, sogenannte Geodatenbanksysteme, etabliert (vgl. BRINKHOFF, 2008). Für diese sind drei Standards maßgeblich: ISO SQL/MM, ISO/TC 211 und OGC Simple Feature SQL. An dieser Stelle sei auf CHEN und XIE (2008) verwiesen, die einen Überblick über diese Standards liefern und zudem eine Reihe von Open Source Geodatenbanksystemen beschreiben. Für die vorliegende Arbeit sind in erster Linie Rasterdaten von Bedeutung, da diese im konkreten Fall der entscheidende Faktor für die Performance des Kartendienstes sind. In den folgenden Abschnitten werden Möglichkeiten zur Haltung und zum Management von Rasterdaten aufgezeigt sowie Werkzeuge beschrieben, die die Verarbeitung der Daten im Kontext WebGIS optimieren.

4.3.1 Rasterdaten

Rasterdaten sind im Gegensatz zu Vektordaten wesentlich einfacher strukturiert, da sie nur einen Entitätstypen besitzen – die Rasterzelle (vgl. Bartelme, 2005). In der digitalen Bildverarbeitung werden diese als Pixel bezeichnet. Sie unterteilen den Datenraum in quadratische oder rechteckige Teilflächen. Ein Raster kann aufgrund seiner homogenen Struktur durch eine zweidimensionale Matrix beschrieben werden (vgl. BURGER und BURGE, 2009).

Folgende Abbildung veranschaulicht das Koordinatensystem eines Rasters:

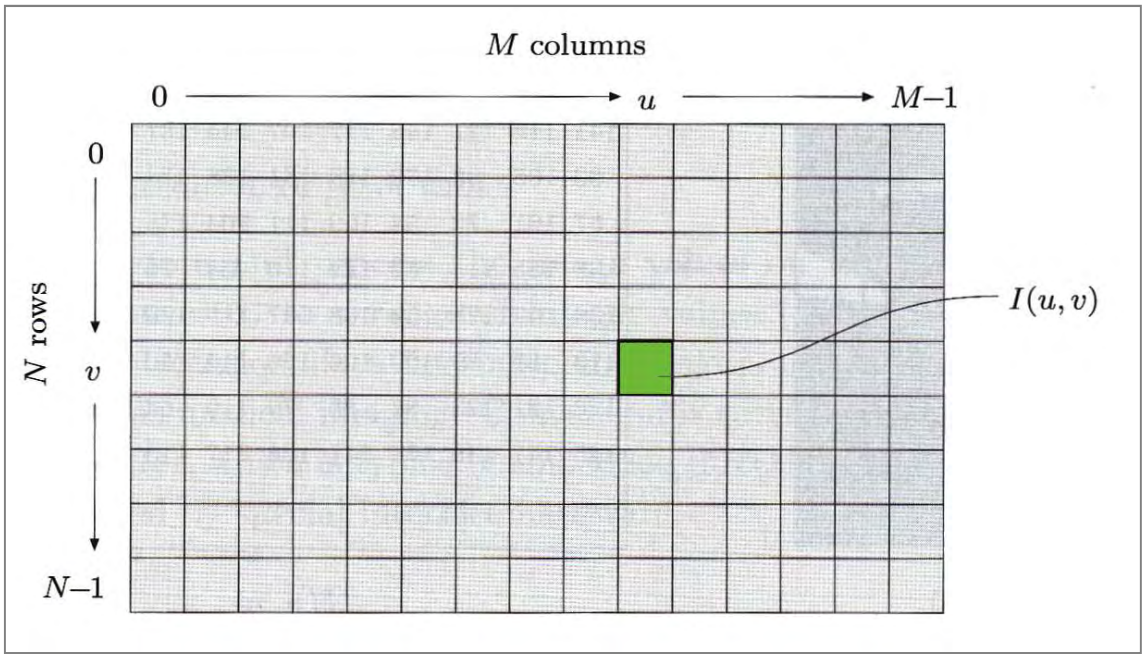


Abb. 21: Bildkoordinatensystem (BURGER und BURGE, 2009)

Der Ursprung des Rasterkoordinatensystems liegt bei (0/0) in der linken oberen Ecke. Diese Konvention hat sich in der digitalen Bildverarbeitung etabliert (vgl. BURGER und BURGE, 2009). Ein Raster mit der Ausdehnung *M* x *N* besitzt die maximale Anzahl von Spalten *M* – 1 und die maximale Anzahl von Zeilen *N* – 1. Eine Rasterzelle wird durch ihre Koordinaten I(u, v) eindeutig identifiziert. Die Informationen (Pixelwerte), die eine Rasterzelle enthalten kann, hängen vom Datentyp bzw. dem Farbmodell des Rasters ab. Sie sind fast immer ein binärer Ausdruck der Länge *k* und können somit 2^k verschiedene Werte einnehmen. *k* wird als Bit-Tiefe oder nur als Tiefe des Bildes bezeichnet (vgl. BURGER und BURGE, 2009).

In einem 8-bit RGB Raster werden beispielsweise die Komponenten für die Farben rot, grün und blau jeweils als eigener Wert zwischen 0 und 255 pro Pixel gespeichert. Das

Raster besitzt demzufolge drei Kanäle und benötigt 3 x 8 = 24 Bits pro Pixel, um die Informationen zu kodieren. RGB Raster, die Transparenz unterstützen, besitzen einen vierten (alpha) Kanal, der die Werte 0 (transparent) oder 1 (opak) einnehmen kann. Das Farbmodell wird dann als RGBA bzw. ARGB bezeichnet und benötigt 32 Bits pro Pixel (vgl. SAMPLE und IOUP, 2010).

Um die Überlegungen zur Datenhaltung von Rasterdaten besser nachvollziehen zu können, muss man sich den Prozessablauf vor Augen führen, bis die fertige Karte den Client erreicht. Wird beispielsweise durch eine Zoomaktion des Clients ein neues Kartenbild vom Server angefordert, so muss dieser zuerst prüfen welche Rasterdaten für den angeforderten Ausschnitt geladen werden müssen. Ist dieser Schritt vollzogen, muss der Kartendienst auf den entsprechenden Teilbereich der Daten zugreifen und ein neues Bild rendern. Ist der Zugriff auf mehrere Quellraster erforderlich, so muss der Kartendienst die jeweiligen Ausschnitte zu einem neuen Kartenbild zusammensetzen, in das festgelegte Ausgabeformat konvertieren und zurück an den Client senden. Die Neuberechnung der Pixelwerte wird als *Resampling* bezeichnet (vgl. KORDUAN und ZEHNER, 2008). Wie lange der Kartendienst für diese Operationen benötigt, hängt nicht nur von der eingesetzten Hardware und der Netzwerkbandbreite, sondern entscheidend von der Organisation bzw. den Formaten der Rasterdaten und deren Komprimierung ab. In den folgenden Abschnitten werden die gängigsten Dateiformate für den WebGIS Bereich vorgestellt, die Verwendung von Geodatenbanksystemen für Rasterdaten skizziert und Optimierungsmöglichkeiten aufgezeigt.

4.3.1.1 Dateibasierte Speicherung

Rasterdaten in ihrer originären Form liegen im Normalfall als unkomprimierte Einzeldateien vor. Je nach Verwendungszweck werden die Raster anschließend komprimiert, um zum einen Speicherplatz zu sparen und zum anderen die Zugriffs- und Übertragungszeiten zu verkürzen. Besitzt beispielsweise ein RGB Raster eine Größe von 5000 x 5000 Pixel, so beläuft sich der Speicherbedarf unkomprimiert auf 5000 x 5000 x 3 Byte = 75000000 Byte = 75 Megabyte). Generell kann in verlustfreie und verlustbehaftete Kompression unterschieden werden. Meist lassen sich mit verlustbehafteten Kompressionsverfahren, z.B. der Diskreten Kosinus-Transformation (DCT), höhere Kompressionsraten erzielen, da Informationen aus dem Bild bzw. Raster entfernt werden (vgl. FISCHER, 2002). Allerdings ist dieser Informationsverlust oftmals für das menschliche

Auge nicht zu erkennen. Somit muss auch bei der Wahl des Kompressionsalgorithmus der Verwendungszweck des Bildes beachtet werden.

Bei der Verwendung von Rasterdaten in einem GIS ist der Raumbezug des Rasters von essentieller Bedeutung. Dieser Raumbezug kann zum einen als Metadaten in den Header des Bildes (z.B. GeoTIFF oder JPEG2000) oder in eine zusätzliche Datei geschrieben werden. Letztere Variante ist die geläufigste. Die Datei mit den Raumbezugsinformationen wird als *Worldfile* bezeichnet. Dieses Worldfile besitzt den gleichen Dateinamen wie das zugehörige Bild. Lediglich die Dateierweiterung ist unterschiedlich. Sie setzt sich aus dem ersten und letzten Buchstaben der Dateierweiterung des Bildes und dem Buchstaben „w“ zusammen. Das Worldfile einer TIFF-Datei hat die Erweiterung *.tfw, die einer PNG-Datei *.pgw (vgl. KORDUAN und ZEHNER, 2008).

4.3.1.1.1 Tagged Image File Format (TIFF)

Das Tagged Image File Format (TIFF) ist eines der am weitesten verbreiteten Rasterdatenformate. Es wurde 1986 von Aldus Corporation entwickelt und die aktuelle Version 6.0 existiert seit 1992. Im Moment hält Adobe Systems die TIFF Spezifikation (vgl. BURGER und BURGE, 2009). Das Format unterstützt eine Vielzahl von Farbmodellen. Dazu zählen RGB, CMYK, Schwarz-Weiß-Daten, Graustufen und Lab-Dateien (vgl. FISCHER, 2002). Eine TIFF-Datei unterstützt Lauflängenkodierung, LZW, CCITT und JPEG Kompression. Aufgrund der Architektur ist das Format sehr flexibel. Die folgende Abbildung zeigt die Architektur des Formats:

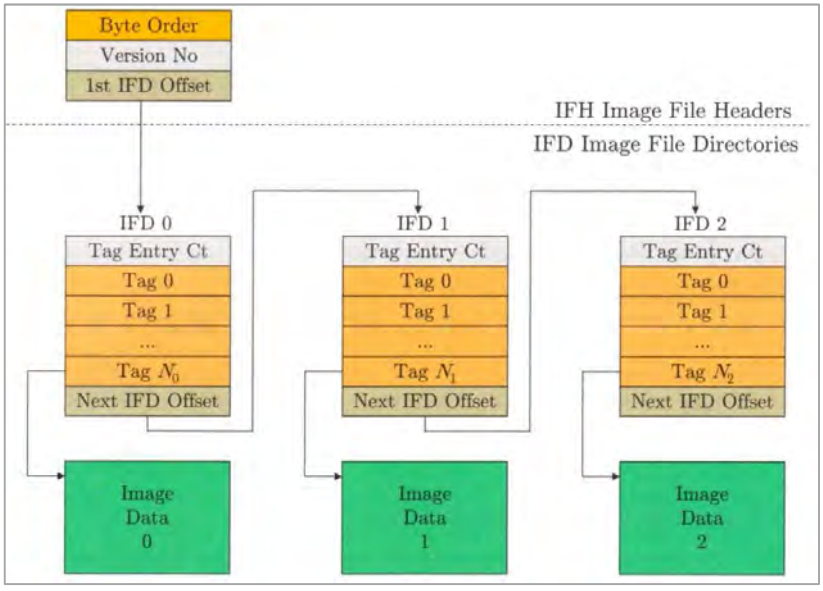


Abb. 22: Architektur einer TIFF-Datei bei drei Bildern (BURGER und BURGE, 2009)

Ein TIFF ist in drei Teile gegliedert. Es existiert der *Image File Header (IFH)*, der Informationen über die Version, die Byte-Ordnung und über das *Image File Directory (IFD)* enthält. Letzteres enthält Felder und Tags. Der dritte Teil besteht aus den eigentlichen Pixelwerten. Eine TIFF Datei kann mehrere IFDs enthalten und damit auch mehrere Bilder (vgl. Fischer, 2002). Die Flexibilität des Formats ist begründet durch die Möglichkeit, eigene Tags definieren zu können. Allerdings stellt dies gleichzeitig auch eine der Schwächen des Formats dar, da nicht jede Software benutzerdefinierte Tags erkennen kann (vgl. BURGER und BURGE, 2009).

Die eigentlichen Bilddaten können im Normalfall in zwei Varianten im TIFF organisiert sein. Sie können einerseits streifenweise angeordnet sein. Andererseits besteht die Möglichkeit die Daten in Kacheln zu speichern. Die Streifen (Strips) stellen eine bzw. mehrere zusammenhängende Zeile(n) von Pixelwerten dar. Aufgrund dieser eindimensionalen Struktur kann darauf leicht zugegriffen werden. Ein weiterer Vorteil dieser Datenorganisation zeigt sich bei großen TIFF Dateien, die viel physikalischen Speicherplatz belegen. Hat das System nicht genügend Arbeitsspeicher zur Verfügung, um eine TIFF Datei vollständig auf einmal zu laden, so erfolgt das Öffnen der Datei „streifenweise“ (vgl. Fischer, 2002). Die Variante, Bilddaten im TIFF auch in Kacheln (Tiles) anordnen zu können, bringt vor allem für die Verwendung in GI-Systemen Vorteile, wenn unterschiedliche Maßstabsbereiche dargestellt werden sollen. Ausführlichst werden Strips und Tiles von MURRAY und VANRYPER (1996) beschrieben.

Unkomprimierte TIFF Dateien stellen oftmals die Rasterdatenbasis in GI-Systemen dar. Beispielsweise werden Digitale Orthophotos von den Vermessungsverwaltungen in diesem Format ausgegeben.

4.3.1.1.2 JPEG/JFIF

Die JPEG Kompression wurde von der Joint Photographic Experts Group (JPEG) gegen Ende der 1980er Jahre entwickelt und ist seit 1990 ein ISO-Standard, der als ISO-10918 bekannt ist. Dabei handelt es sich um ein verlustbehaftetes Verfahren, das auf der DCT basiert und 8-bit Tiefe pro Bildpunkt erlaubt (vgl. STRUTZ, 2009). Eigentlich ist JPEG kein Dateiformat, sondern ein Standard zur Einzelbildkompression. Was normalerweise unter einer JPEG Datei verstanden wird, ist das *JPEG File Interchange Format (JFIF).*

JPEG ermöglicht hohe Kompressionsraten bis zum Faktor 1:25. Allerdings verringert sich dadurch die Bildqualität, da diese indirekt-proportional zum Kompressionsgrad ist. Ab einer mittleren Kompressionsstufe sind die Verluste im Bild mit dem menschlichen Auge wahrnehmbar (vgl. FISCHER, 2002). JPEG unterstützt graustufige Bilder und die Farbräume RGB und CMYK. Die JPEG Kompressionsalgorithmen sind äußerst komplex und werden detailliert bei STRUTZ (2009) erläutert. Des Weiteren existiert die waveletbasierte, verlustbehaftete Kompression JPEG-2000 und auch eine verlustfreie JPEG Kompression (JPEG-LS), die allerdings nur selten Verwendung findet (vgl. FISCHER, 2002).

4.3.1.1.3 Graphics Interchange Format (GIF)

Die 1987 von CompuServe veröffentlichte GIF Spezifikation stellt eines der am häufigsten verwendeten Bildformate im Internet dar. Eine GIF Datei ist auf maximal 256 Farben bei 8-bit Tiefe beschränkt und wird immer mit dem patentierten LZW-Verfahren komprimiert. Dabei stellt die beschränkte Farbanzahl einen limitierenden Faktor dar. Für Bilder deren Farbübergänge fein sind, wie beispielsweise Digitale Orthophotos, ist das Format ungeeignet (vgl. FISCHER, 2002). Es ist eher für Bilder geeignet, die großflächig die gleiche Farbe besitzen (vgl. BURGER und BURGE, 2009).

4.3.1.1.4 Portable Network Graphics (PNG)

Das Format Portable Network Graphics (PNG) wurde als Ersatz für das GIF Format entwickelt als es zu lizenzrechtlichen Problemen bei dessen LZW-Kompression kam. Es wurde speziell für die Anwendung im Internet konzipiert und 1996 erstmals spezifiziert. PNG unterstützt drei Bildtypen bzw. Farbräume: RGB mit bis zu 16-bit pro Pixel, grauskalierte Bilder mit bis zu 16-bit pro Pixel und indizierte Bilder bis 256 Farben (vgl. BURGER und BURGE, 2009). Darüber hinaus ist ein Alpha-Kanal vorhanden. PNG Dateien sind immer komprimiert. Dabei wird die *Deflate Compression* angewendet, eine

Variante des LZ77-Algorithmus, die eine verlustfreie Kompression ermöglicht. Wie das GIF-Format, so unterstützt PNG auch das sogenannte *Interlacing*, allerdings in einer abgewandelten Form. Dabei werden durch das *Adam7*-Verfahren die Bilddaten in sieben Durchgängen, beginnend mit den ungeraden Zeilen, schubweise übertragen. Dies ermöglicht einen wesentlich schnelleren Bildaufbau (vgl. FISCHER, 2002).

4.3.1.1.5 Windows Bitmap (BMP)

Das BMP ist das Standardformat für Bilddateien im Windows Betriebssystem. Es unterstützt die Farbmodelle Schwarz/Weiß, Graustufen, indizierte Farben und RGB. Es sind entweder ein oder drei Kanäle möglich. Ein Alpha Kanal existiert nicht. Das BMP Format ist im Normalfall unkomprimiert. Allerdings wird auch eine Variante der Lauflängenkodierung unterstützt (vgl. FISCHER, 2002). Letztendlich ist das Format sehr unflexibel.

4.3.1.2 Geodatenbanken

Neben der dateibasierten Speicherung können Rasterdaten auch in objektrelationalen Datenbanksystemen gespeichert werden. Häufig werden Rasterdaten als *Binary Large Objects (BLOB)* gespeichert. Eine Abfrage einzelner Rasterzellen auf Datenbankebene ist dabei allerdings nicht möglich. Auswertungen müssen mittels zusätzlicher Anwendungen erfolgen (vgl. BRINKHOFF, 2008). Es existieren kaum Systeme, die Rasterdaten verwalten können und auch räumliche Analysefunktionen dafür bereitstellen. Zu nennen sind vier Softwareprodukte: rasdaman, Oracle Spatial, ESRI ArcSDE und PostGIS Raster.

ArcSDE von ESRI und *rasdaman* von rasdaman GmbH sind beide kommerzielle Middlewareprodukte, wobei letzteres seit kurzem auch als Open Source Variante zur Verfügung steht. Aufgrund der fehlenden Unterstützung für das Windows Betriebssystem wird auf rasdaman nicht näher eingegangen. Näheres zu rasdaman und Rasterdatenbanken gibt BAUMANN (2009). Oracle Spatial hat mit *Georaster* die Möglichkeit geschaffen, georeferenzierte Rasterdaten zu speichern und zu analysieren. Es wird auf die Ausführungen von BRINKHOFF (2008) verwiesen.

Einen anderen Ansatz verfolgt PostGIS Raster, vormals als PostGIS wktraster bezeichnet. PostgreSQL ist ein Open Source objektrelationales Datenbanksystem, das sich mit der Open Source Erweiterung PostGIS zum Standard Geodatenbankmanagementsystem für die meisten Open Source GIS Anwendungen entwickelte (vgl. RAMSEY, 2007).

PostGIS Raster befindet sich derzeit noch in Entwicklung und wird ab PostGIS 2.0 fest darin implementiert sein (vgl. http://trac.osgeo.org/postgis/wiki/WKTRaster). Ziel des PostGIS Raster Projekts ist die Implementierung eines eigenen *Raster* Datentyp in PostGIS, der mit dem *Geometry* Datentyp vergleichbar ist. Darüber hinaus unterstützt GDAL ab Version 1.6.0 PostGIS Raster, so dass Rasterdaten, die in PostgresSQL / PostGIS organisiert sind auch mit MapServer UMN visualisiert werden können. Die aktuelle Version von PostGIS Raster ist 0.1.6d. Diese kann zusammen mit einer kompletten PostgreSQL / PostGIS Installation unter http://www.postgis.org/download/windows/experimental.php heruntergeladen werden. Zusätzlich muss Python ab Version 2.5 verwendet werden. Die Möglichkeit Rasterdaten innerhalb (in-db) oder außerhalb (out-db) der Datenbank organisieren zu können, macht das System äußerst flexibel.

Der Ablauf, Rasterdaten mit PostgreSQL / PostGIS und MapServer UMN nutzen zu können, ist folgender:

Abb. 23: Schema PostGIS Raster Mapfile Integration (eigener Entwurf)

Im Mapfile ist der Layer als TYPE Raster und im DATA Tag ist die Datenbankverbindung anzugeben. Folgendes Codebeispiel zeigt die Mapfile Konfiguration für einen PostGIS Raster Layer. In der Datenbank *RasterDB* hat das Raster den Namen *testraster* und befindet sich im Datenbankschema *raster*:

```
LAYER
  NAME "Testraster"
```

```
    TYPE Raster
    STATUS OFF
    DATA "PG:host=localhost port=5432 dbname='RasterDB'
          user='postgres' password='xxxxxxxx'
          schema='raster' table='testraster'"
     CLASSITEM "[pixel]"
      CLASS
        EXPRESSION  ([pixel] = 4)
        STYLE
         COLOR 153 255 179
        END
      END
END
```

Die Informationen zur MapServer UMN Integration von PostGIS Raster wurden von der Seite http://trac.osgeo.org/postgis/ticket/497 bezogen. Die Dokumentation im Betastadium zu PostGIS Raster findet sich unter http://trac.osgeo.org/postgis/wiki/WKTRaster/Documentation01.

4.3.2 Optimierungsmöglichkeiten des Datenzugriffs

Unabhängig vom Datenformat, müssen noch weitere technische Optionen im Bereich der Datenorganisation herangezogen werden, um ein optimales Ergebnis im Hinblick auf die Performance erzielen zu können. Sequentielles Lesen der Eingangsdaten hat zur Folge, dass unter Umständen viel Zeit verloren geht bis der Kartendienst die wirklich benötigten Datensätze bzw. Pixelwerte gefunden hat und diese zu einem Ausgabebild rendern kann (vgl. KORDUAN und ZEHNER, 2008). Eine Strategie, diese Unzulänglichkeit zu umgehen, bildet die Verwendung räumlicher Indizes, durch die auf die gesuchten Objekte schneller zugegriffen werden kann (vgl. BARTELME, 2005).

4.3.2.1 Indizierung

In MapServer UMN werden die Indizes von nicht in Geodatenbanksystemen organisierten Daten als *Tileindizes* bezeichnet. Ein Tileindex ist ein ESRI Shapefile, das mehrere Quelldatenbestände zu einem einzigen Layer kombiniert. Dabei wird das Prinzip der *Mosaikierung* bzw. *Kachelung* heran gezogen. Das Shapefile besteht aus Polygonen, die der räumlichen Ausdehnung der einzelnen Quelldaten entsprechen und deckt somit das ganze Gebiet aller zusammengefassten Quelldaten ab. Jedes Polygon besitzt dabei den Pfad bzw. den Dateinamen seines Quelllayers als Attribut und referenziert diesen somit.

Die nachfolgende Abbildung verdeutlicht das Prinzip eines Tileindex:

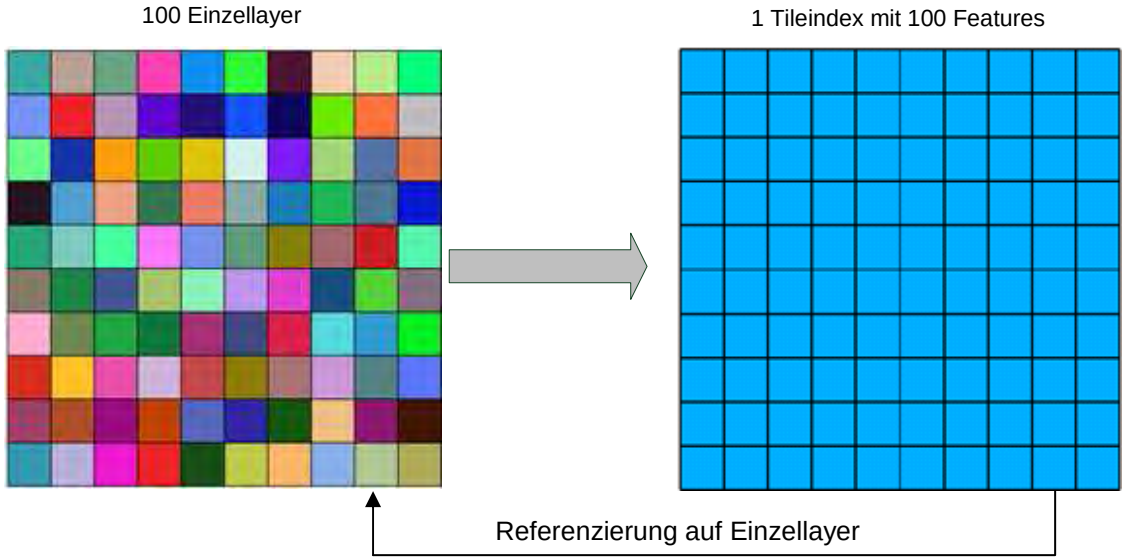


Abb. 24: Prinzip des Tileindex (eigener Entwurf)

Das Beispiel zeigt eine idealisierte Form des Tileindex, da die Einzellayer aneinander grenzen und zudem quadratisch sind. Dies ist jedoch nicht zwingend erforderlich. Die Referenzierung mit einem Tileindex kann auch erfolgen, wenn die Einzellayer ungleichförmige Ausdehnungen besitzen und nicht aneinander grenzen.

Tileindizes können die Performance von MapServer UMN steigern. Dies liegt zum einen daran, dass im Mapfile nur noch der Tileindex als LAYER angegeben werden muss, anstelle 100 einzelner LAYER Elemente wie im obigen Beispiel. Zum anderen öffnet MapServer UMN bei einem Request zuerst den Tileindex und stellt daraufhin fest, welche Einzellayer für die gerade angeforderte Karte benötigt werden. Nur diese werden anschließend noch geladen. Allerdings kann die Verwendung eines Tileindex auch einen Performanceverlust mit sich bringen, in der Regel dann wenn viele Einzellayer für das Rendering der Karte benötigt werden. In diesem Fall kann es performanter sein, einen „großen“ Layer anstelle vieler „kleiner“ zu laden (vgl. HOSTGIS, 2007).

Tileindizes können sowohl für Raster- als auch für Vektordaten erstellt werden. Ein Tileindex kann mit dem GDAL Tool *gdaltindex* für Rasterdaten und für Vektordaten mit *ogrtindex* erzeugt werden. Die Verwendung beider Tools ist vom Prinzip her gleich:

```
gdaltindex [Name des Tileindex].shp [Pfad zum Verzeichnis
mit den Einzelbildern]/*.[Dateierweiterung der
Einzellayer]
```

Ein Beispiel um für alle im Verzeichnis c:\test enthaltenen TIFF-Dateien einen Tileindex zu erstellen, könnte wie folgt aussehen:

```
gdaltindex tileexample.shp c:/test/*.tif
```

Im Mapfile wird ein Tileindex im LAYER Element angegeben, z.B.:

```
LAYER
  NAME "DOP"
  TYPE Raster
  STATUS OFF
  TILEINDEX "c:/DOP/dop.shp"
END
```

TIFF-Dateien können seit der Version 6.0 interne Tiles enthalten (vgl. FISCHER, 2002). Eine TIFF-Datei besteht dann aus vielen Kacheln. Diese können wiederum leicht mit einem GDAL Werkzeug (gdal_translate) erstellt werden (HOSTGIS, 2008):

```
gdal_translate -co TILED=YES original.tif tiled.tif
```

Eine weitere Möglichkeit Rasterdaten zu indizieren, ist die Verwendung eines *Virtual Datasets (VRT)*. Dabei handelt es sich um ein GDAL-spezifisches Format, das die Metadaten und Referenzierung einer oder mehrerer Quelllayer in eine XML Datei mit der Erweiterung *.vrt schreibt. Ein VRT Beispiel für zwei Rasterdateien zeigt folgende Abbildung:

```
<VRTDataset rasterXSize="4476" rasterYSize="4116">
  <GeoTransform> 4.4513844411800001e+006, 2.5000000000000000e+001, 0, 5.4101172941199997e+006, 0,-2.5000000000000000e+001
  </GeoTransform>
  <VRTRasterBand dataType="Byte" band="1">
    <NoDataValue>0</NoDataValue>
    <ColorInterp>Gray</ColorInterp>
    <ComplexSource>
      <SourceFilename relativeToVRT="1">raster01.tif</SourceFilename>
      <SourceBand>1</SourceBand>
      <SourceProperties RasterXSize="540" RasterYSize="522" DataType="Byte" BlockXSize="540" BlockYSize="15"/>
      <SrcRect xOff="0" yOff="0" xSize="540" ySize="522"/>
      <DstRect xOff="0" yOff="0" xSize="540" ySize="522"/>
      <NODATA>0</NODATA>
    </ComplexSource>
    <ComplexSource>
      <SourceFilename relativeToVRT="1">raster02.tif</SourceFilename>
      <SourceBand>1</SourceBand>
      <SourceProperties RasterXSize="502" RasterYSize="486" DataType="Byte" BlockXSize="502" BlockYSize="16"/>
      <SrcRect xOff="0" yOff="0" xSize="502" ySize="486"/>
      <DstRect xOff="3974" yOff="3630" xSize="502" ySize="486"/>
      <NODATA>0</NODATA>
    </ComplexSource>
  </VRTRasterBand>
</VRTDataset>
```

Abb. 25: Beispiel eines GDAL Virtual Datasets (eigener Entwurf)

Ein Virtual Dataset kann mit dem GDAL Tool *gdalbuilvrt* erstellt werden:

```
gdalbuilvrt vrtexample.vrt c:/test/*.tif
```

Um ein Virtual Dataset in einem Mapfile einzubinden, muss einfach das DATA Tag des Layers auf die VRT Datei verweisen, z.B.:

```
LAYER
  NAME "DOP"
  TYPE Raster
  STATUS OFF
  DATA "c:/DOP/dop.vrt"
END
```

Weitere Informationen zu den GDAL Virtual Datasets sind unter http://www.gdal.org/gdal_vrttut.html zu finden.

In PostgreSQL / PostGIS organisierte Daten können mit dem GiST (Generalized Search Tree Index) indiziert werden. Es handelt sich dabei um eine auf den R-Baum aufbauende Indexstruktur (vgl. http://www.sai.msu.su/~megera/postgres/gist/). Dieser räumliche Index kann sowohl auf Vektor- als auch auf Rasterdaten angewendet werden.

4.3.2.2 Overviews

Neben der Indizierung der Rasterdaten, besteht noch die Möglichkeit durch die Verwendung von Bildpyramiden die Performance von MapServer UMN zu verbessern. Die Bildpyramiden werden auch als Overviews bezeichnet. Bildpyramiden stellen das Quellraster in unterschiedlichen Auflösungen dar. Dieses muss dazu resampled werden. Die Auflösung einer Ebene der Bildpyramide ist dabei immer geringer als das Originalbild, in der Regel wird sie halbiert (vgl. BRINKHOFF, 2008). Die folgende Abbildung verdeutlicht dies:

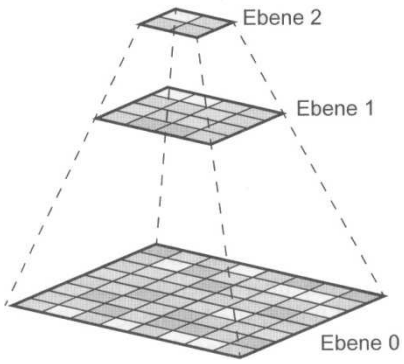


Abb. 26: Bildpyramide (BRINKHOFF, 2008)

Bildpyramiden bzw. Overviews werden mit dem GDAL Tool *gdaladdo* erzeugt. Dabei muss beachtet werden, dass interne oder externe Overviews erstellt werden können. Erstere werden in die Originaldatei geschrieben, sofern dies von dem entsprechenden Format unterstützt wird, z.B. TIFF. Bei externen Overviews wird eine Datei mit dem

gleichen Dateinamen des Quellrasters und der Erweiterung *.ovr generiert. Folgendes Beispiel zeigt die Erzeugung externer Overviews für ein Raster:

```
gdaladdo -ro D:/raster01.tif 2 4 8 16 32 64 128
```

Um interne Overviews zu erstellen, muss die Option –ro weggelassen werden. Es kann auch der Resampling Algorithmus festgelegt werden. Details zur Syntax von gdaladdo werden auf http://www.gdal.org/gdaladdo.html angeführt. Für die Verwendung von Overviews in MapServer UMN muss nichts weiter beachtet werden, da der GDAL Treiber automatisch erkennt, ob Overviews vorhanden sind und diese dann entsprechend verwendet.

4.4 Performance

Wie eingangs geschildert, stellt die Perfomance eines WebGIS Systems einen nicht zu vernachlässigenden Aspekt dar, der schon beim Aufbau eines Portals zu berücksichtigen ist. Können in lokalen Netzwerken noch präzise die Anwenderzahlen abgeschätzt werden, ist es bei öffentlich exponierten Diensten relativ schwer, die Zugriffszahlen vorherzusagen. Nach Inbetriebnahme eines WebGIS Portals ist es oftmals nicht leicht möglich, zuvor festgelegte Strukturen und Konfigurationen ohne Server-Ausfallzeiten zu ändern. Große Datenmengen aufgrund von Fehlentscheidungen oder falschen Annahmen umzuorganisieren bzw. zu konvertieren kann viel Zeit und Ressourcen in Anspruch nehmen. Daher ist eine vorab durchgeführte Performanceanalyse bzw. Teststellung unerlässlich (vgl. KORDUAN und ZEHNER, 2008).

Unter Performance ist die Effizienz und Effektivität von IT-Systemen bzw. Applikationen zu verstehen. Unterschieden werden drei Arten: Client-, Server und Netzwerkperformance. Alle drei gemeinsam spiegeln die allgemeine Systemperformance wider. Bestimmt wird die Performance durch die Systemauslastung und die Geschwindigkeit der einzelnen Komponenten (vgl. PENG und TSOU, 2003).

Clientseitig kann Performance, als die vom Anwender wahrgenommene Antwortzeit des Servers interpretiert werden. Die Systemauslastung auf Clients ist für gewöhnlich gering. Allerdings wird versucht, möglichst viele Arbeitsschritte bereits auf dem Client ablaufen zu lassen, um damit nicht unnötig den Server belasten zu müssen. Ein Beispiel dafür sind Bemaßungswerkzeuge. Manche WebGIS Applikationen nutzen dafür Plugins, Applets oder ActiveX-Steuerelemente, die aber die Clientperformance negativ beeinflussen können. Selbst dann, wenn der Client seine Ressourcen optimal nutzt, ist dies keine Garantie für ein schnelles WebGIS Portal. Letztendlich muss versucht werden,

den Datentransfer zwischen Client und Server auf das Nötigste zu reduzieren. Die bereits beschriebene Ajax Technologie ist ein Schritt in diese Richtung. Eine weitere Möglichkeit clientseitig die Performance zu verbessern, ist das Zwischenspeichern von Daten auf dem Client, was als *Caching* bezeichnet wird (vgl. KORDUAN und ZEHNER, 2008). Oftmals werden auch Techniken angewendet, um die subjektiv vom Anwender wahrgenommene Performance zu verbessern. Dazu zählt das schrittweise Ausliefern von Daten an den Client, z.B. gekachelte Kartenauslieferung oder auch der Einsatz von Elementen, die den aktuellen Status der Operation anzeigen (vgl. PENG und TSOU, 2003).

Die Hauptlast trägt der Server. Dementsprechend müssen Überlastungen vermieden werden. Diese können nicht nur durch eine große Anzahl von Clientrequests, sondern auch durch rechenintensive Anfragen auftreten. Die Serverperformance wird zum einen durch die eingesetzte Hardware, zum anderen durch die Konfiguration des Servers selbst beeinflusst (vgl. PENG und TSOU, 2003). Im vorliegenden Fall ist damit die Optimierung von MapServer UMN zu verstehen. Neben der Wahl der Datenformate und der Zugriffsoptimierung durch Indizes, kann die Performance durch die Verwendung von *FastCGI* verbessert werden.

Bei FastCGI (FCGI) handelt es sich quasi um eine CGI-Schnittstelle mit Erweiterungen. Sie besitzt gegenüber CGI zwei entscheidende Vorteile. FastCGI Prozesse werden nach dem Abarbeiten eines Requests nicht beendet, sondern warten auf weitere Requests. Anstatt Umgebungsvariablen auf Betriebssystemebene zu verwenden, nutzt FastCGI die Kommunikation per TCP zwischen Webserver und dem FastCGI Programm. Dieser Umstand ermöglicht es, das FastCGI Programm vom Webserver zu trennen. Des Weiteren unterstützt FastCGI Single- und Multithreading. Bei ersterem verwaltet der Webserver einen Prozesspool, um Clientanfragen abzuarbeiten. Multithreading Applikationen können mehrere Clientanfragen gleichzeitig in einem einzigen Prozess bewältigen (vgl. OPEN MARKET, 1996). Mit FastCGI kann MapServer UMN entscheidende Performancegewinne erzielen. Darauf wird in der MapServer UMN Dokumentation ausdrücklich hingewiesen.

Die Performance im Netzwerk stellt gewöhnlich den sprichwörtlichen Flaschenhals von WebGIS Applikationen dar. Entgegnen kann man diesem Problem mit möglichst geringen Datenvolumina, die zwischen Client und Server übertragen werden müssen. Obwohl die Datenübertragungsgeschwindigkeit im Internet rasant zugenommen hat, darf

die Berücksichtigung der Netzwerkperformance nicht vernachlässigt werden. Diese ist allerdings mit einer einzigen Messung nicht zu erfassen, da sie vom Netzwerktraffic und somit von der Zeit abhängig ist. Als Kennzahlen dienen der *Durchsatz* und die *Latenzzeit*. Unter dem Durchsatz versteht man die Anzahl an Bits, die pro Zeiteinheit übertragen werden. Die Latenzzeit kann mit der *Antwortzeit* verglichen werden. Sie kennzeichnet die Zeitspanne, die zwischen Clientrequest und Serverantwort vergeht und beinhaltet damit eine Client-, eine Netzwerk- und eine Serverkomponente. Im Zusammenhang mit WebGIS Applikationen wird die Kennzahl Durchsatz mit Karten pro Zeiteinheit gleichgesetzt (vgl. PENG und TSOU, 2003).

5 MapServer UMN WMS Benchmark

Nachdem die technischen Grundlagen für den Aufbau eines WebGIS Portals dargelegt wurden, soll in einem nächsten Schritt die optimale Konfiguration zur Visualisierung von Rasterdaten mit MapServer UMN ermittelt werden. Um dies herauszufinden, müssen die Daten in die zu testenden Formate konvertiert und in ein MapServer UMN Mapfile integriert werden. Nach einem festgelegten Schema werden die Zeiten registriert, die der Kartendienst benötigt, bis die Karten erzeugt sind.

Dabei wird sich nach der Vorgehensweise des FOSS4G WMS Benchmark (http://wiki.osgeo.org/wiki/FOSS4G_Benchmark) gerichtet. Der Benchmark wird jährlich seit 2007 durchgeführt und hat zum Ziel, die Performance verschiedener WMS Serversysteme zu vergleichen und diese damit zu verbessern. Begonnen wurde der Vergleich mit MapServer UMN und Geoserver. Mittlerweile wurden auch die Kartendienste Cadcorp GeognoSIS, Constellation SDI, Erdas Apollo, Mapnik, Oracle Mapviewer und QGIS MapServer getestet.

5.1 Testdaten

Für die Durchführung der Tests wurden nur Rasterdaten verwendet, obgleich für die Umsetzung des Portals auch Vektordaten visualisiert werden sollen. Dies ist zum einen darauf zurückzuführen, dass sich ZETTEL (2007) bereits mit der Vektordatenverarbeitung von MapServer UMN befasst hat und zum anderen, dass für den vorliegenden Fall die Rasterdatenverarbeitung den entscheidenden Faktor für die Performance des Systems darstellt.

Da die zum Testen zur Verfügung gestellten Rasterdaten aus der Straßenbefahrung nicht zur Veröffentlichung gedacht sind, werden nur die Bildeigenschaften und die Ergebnisse präsentiert. Zusätzlich wurden vom Landratsamt Miesbach Rasterdaten bereitgestellt. Die Daten wurden aus ihren ursprünglichen Formaten in folgende Dateiformate konvertiert: JPEG/JFIF, PNG, GIF, JPEG2000. Die Konvertierungen nach JPEG/JFIF, PNG und GIF wurden mit der für private Zwecke freien Software *IrfanView* (http://www.irfanview.de/) durchgeführt. Die JPEG2000-Dateien wurden mit dem *ER Mapper ECW JPEG2000 Compressor* von Earth Ressource Mapping Ltd. konvertiert. Die gekachelten TIFF-Dateien sind mit dem GDAL Tool gdal_translate erstellt worden.

Alle Testdaten wurden auch in die PostgreSQL Datenbank als PostGIS Raster importiert. Allerdings konnte mit MapServer UMN nur die Topographische Karte visualisiert

werden. Dies ist vermutlich auf die verwendete GDAL Version zurückzuführen, die PostGIS Raster nur bei Verwendung einer Farbtiefe von 8-bit visualisieren kann. Deshalb wurde PostGIS Raster bei den anderen Testdaten nicht berücksichtigt. Die Größe der Rastertabelle wurde mit folgender SQL Abfrage ermittelt:

```
SELECT
pg_size_pretty(pg_total_relation_size('SCHEMANAME.TABELLEN
NAME));
```

5.1.1 Rasterdaten aus der Straßenbefahrung

Aus der Straßenbefahrung liegt ein hoch aufgelöstes, mosaikiertes Digitales Orthophoto vor, das eine ursprüngliche Dateigröße von ca. 390 MB aufweist. Um das Ausgangsraster in JPEG2000 konvertieren zu können, wurde es resampled, da sonst eine Lizenz für ER Mapper erforderlich gewesen wäre. Folgende Tabelle listet die Details zu den einzelnen Formaten auf:

Format	Dateigröße [kB]	Bildgröße	Pixelgröße	Blockgröße
TIFF	388936	10048, 39605	0,002 / -0,002	10048x1
TILED TIFF	396851	10048, 39605	0,002 / -0,002	256x256
JPEG / JFIF	16025	10048, 39605	0,002 / -0,002	10048x1
PNG	86271	10048, 39605	0,002 / -0,002	10048x1
GIF	99358	10048, 39605	0,002 / -0,002	10048x1
JPEG2000	5753	6658, 26243	0,003 / -0,003	3 x 6658 x 1

Tab. 11: Eigenschaften der Rasterdaten aus der Straßenbefahrung (eigener Entwurf)

5.1.2 Vergleichsdaten

Vom Landratsamt Miesbach wurden Rasterdaten in einem 10 x 10 km großen Ausschnitt für die vorliegende Arbeit zur Verfügung gestellt. Dabei handelt es sich um Digitale Orthophotos mit 20 cm Bodenauflösung und um die Topographische Karte 1:50000 der Bayerischen Vermessungsverwaltung.

Nachfolgende Tabelle führt die Eigenschaften der Digitalen Orthophotos auf:

Format	Dateigröße [kB]	OVR Dateigröße [kB]	Bildgröße	Pixelgröße	Blockgröße
TIFF	73248	25934	5000, 5000	0,2 / -0,2	3 x 5000 x 8
TILED TIFF	102733	25934	5000, 5000	0,2 / -0,2	3 x 256 x
JPEG/JFIF	17896	25934	5000, 5000	0,2 / -0,2	3 x 5000 x 1
PNG	37552	25934	5000, 5000	0,2 / -0,2	3 x 5000 x 1
GIF	16252	8656	5000, 5000	0,2 / -0,2	5000 x 1
JPG2000	2999	-	5000, 5000	0,2 / -0,2	3 x 5000 x 1

Tab. 12: Eigenschaften der Digitalen Orthophotos (eigener Entwurf)

Die Eigenschaften der Topographischen Karte sind folgende:

Format	Dateigröße [kB]	OVR Dateigröße [kB]	Bildgröße	Pixelgröße	Blockgröße
TIFF	15643	5503	4000, 4000	2,5 / -2,5	4000 x 2
TILED TIFF	16392	5503	4000, 4000	2,5 / -2,5	256 x 256
JPEG / JFIF	8416	16474	4000, 4000	2,5 / -2,5	3 x 4000 x 1
PNG	1199	5503	4000, 4000	2,5 / -2,5	4000 x 1
GIF	1563	5503	4000, 4000	2,5 / -2,5	4000 x 1
JPG2000	9677	-	4000, 4000	2,5 / -2,5	3 x 4000 x 1
PostGIS Raster	2104	-	4000, 4000	2,5 / -2,5	256 x 256

Tab. 13: Eigenschaften der Topographischen Karte (eigener Entwurf)

Übersichtskarten mit den Testdaten sind im Anhang dieser Arbeit zu finden (Anlagen 3 und 4).

5.2 Testumgebung

Die Tests wurden in einer isolierten, virtualisierten Umgebung in einem 1000 Mbit Netzwerk durchgeführt. Die physische Hardware besteht aus zwei Hewlett Packard Proliant DL 380 G6 Servern mit jeweils zwei vierkernigen Intel Xeon E5540 Prozessoren mit 2,53 GHz Taktfrequenz und 14 GB Arbeitsspeicher, die über einen physischen 24-Port Switch verbunden sind. Als virtuelles Hostbetriebssystem wurde ESXi 4.1 von

vmware verwendet. Der Testclient und die Testserver sind virtuelle Maschinen, die auf den lokalen Festplatten der physischen Server laufen. Als Clientbetriebssystem wurde Windows XP, als Serverbetriebssystem Windows Server 2008 eingesetzt. Alle Maschinen besitzen drei GB Arbeitsspeicher.

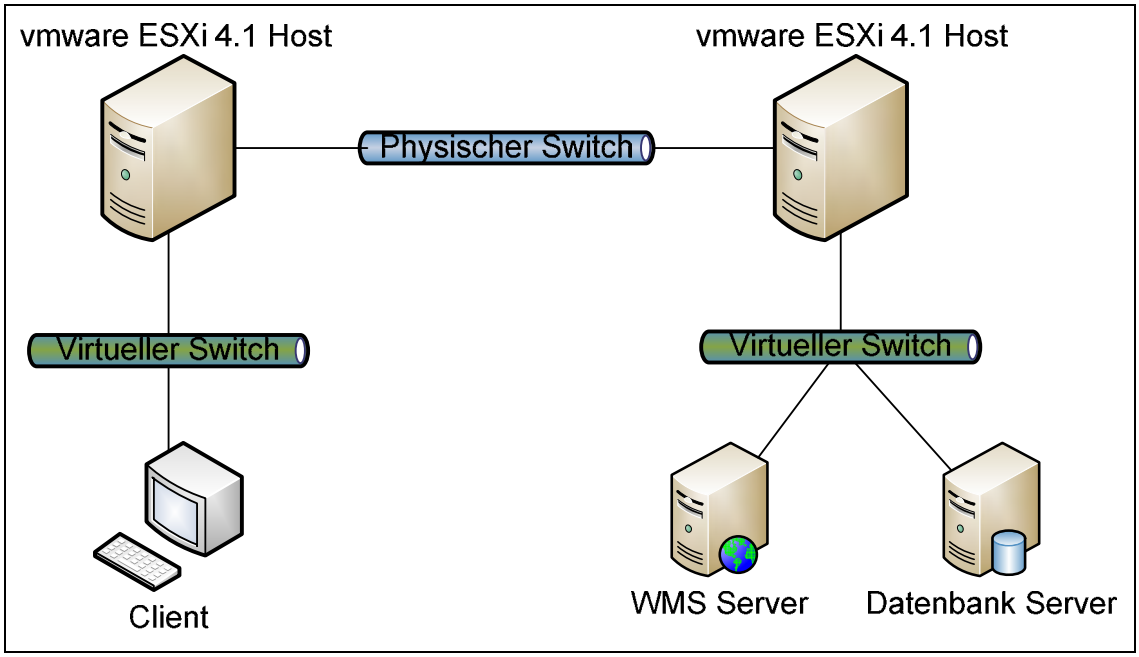


Abb. 27: Schema der Testumgebung (eigener Entwurf)

Um die Performance des WMS Servers exakt messen zu können, wurde auf dem Client die Software *JMeter* des Apache Jakarta Projekts (http://jakarta.apache.org/jmeter/) in der Version 2.4 eingesetzt. Auf dem WMS Server wurde das MS4W Paket mit Apache 2.2.15 und mod_fcgid 2.3.5, MapServer UMN 5.6.3 und GDAL 1.7.1 installiert. Die Versionsinformationen von MapServer UMN lauten:

```
MapServer version 5.6.3 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG
OUTPUT=WBMP OUTPUT=PDF OUTPUT=SWF OUTPUT=SVG SUPPORTS=PROJ
SUPPORTS=AGG SUPPORTS=FREETYPE SUPPORTS=ICONV SUP-
PORTS=FRIBIDI SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT SUP-
PORTS=WFS_SERVER SUPPORTS=WFS_CLIENT SUPPORTS=WCS_SERVER
SUPPORTS=SOS_SERVER SUPPORTS=FASTCGI SUPPORTS=THREADS SUP-
PORTS=GEOS SUPPORTS=RGBA_PNG SUPPORTS=TILECACHE INPUT=JPEG
INPUT=POSTGIS INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE
```

Zum Testen von PostGIS Raster wurde auf dem Datenbankserver PostgreSQL 9.0 mit PostGIS 1.5.2-3 und wktraster 0.1.6d installiert.

5.3 Testdurchführung

Man kann generell zwei Arten von Benchmarks unterscheiden. Zum einen „heiße“, bei denen die Systemzwischenspeicher (Cache) gefüllt sind und zum anderen „kalte“ Benchmarks. Bei letzteren ist das Betriebssystem frisch gestartet – der Cache ist deshalb leer. Beide Szenarien sind nicht realitätsnah, da im Normalfall eine Mischung beider Varianten vorkommt. Für die vorliegende Arbeit wurden heiße Benchmarks durchgeführt, da sie wesentlich praktikabler in der Umsetzung sind und zudem auch von der FOSS4G verwendet werden.

Jeder einzelne Test wird mit 1 (100 Wiederholungen), 10 (20 Wiederholungen), 20 (20 Wiederholungen) und 40 (20 Wiederholungen) parallelen Clientzugriffen durchgeführt, so dass sich pro Test eine Anzahl von 1500 Requests ergeben. Jeder Test wird dreimal nacheinander vollzogen, wobei nur die Werte des dritten Durchlaufs in das Endergebnis einfließen. Um für jeden Testlauf dieselbe räumliche Ausdehnung und Bildgröße vom WMS Server abzurufen, wurde mit dem Python Skript *wms_request.py* von WARMERDAM (2009) eine Konfigurationsdatei für jedes Testgebiet angelegt. Diese Konfigurationsdatei wird JMeter zur Verfügung gestellt, das daraus die Parameter der Bildgröße (WIDTH und HEIGHT) und der räumlichen Ausdehnung (BBOX) für die WMS GetMap Requests entnimmt. Es wurde ferner darauf geachtet, dass in diesen Konfigurationsdateien so viele verschiedene Auswahlmöglichkeiten zur Verfügung stehen, dass der gleiche GetMap Request pro Durchlauf nur einmal ausgeführt werden kann. Die minimale Bildauflösung beträgt 640x480, die maximale 1024x768. Ein Beispieldatensatz dieser Konfigurationsdateien mit Erläuterung sieht wie folgt aus:

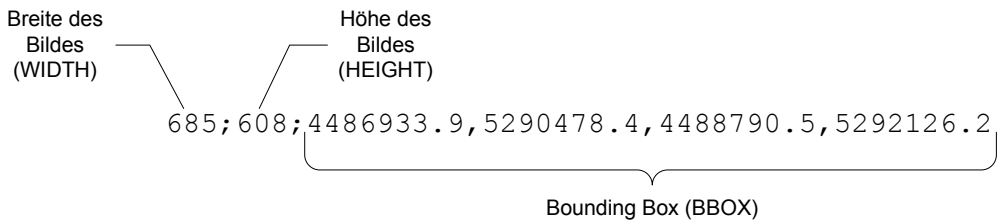


Abb. 28: Beispieldatensatz wms_request.py (eigener Entwurf)

JMeter bietet die Möglichkeit mit dem Konfigurationselement *HTTP Request Defaults*, den statischen Teil des WMS Aufrufs zu definieren. Neben den statischen Parametern werden die Server IP, der Port und der Pfad zur Webapplikation angegeben. Jeder Test besteht aus vier *Thread-Gruppen*, die der Anzahl der Clientzugriffen entsprechen. Jede einzelne dieser Thread-Gruppen besteht aus den Elementen *Schleifen Controller* (Loop

Controller), *HTTP Request* und *CSV Einstellungen* (CSV Dataset Config). Mit Hilfe von *Listener* Elementen können die Ergebnisse des Tests zusammengefasst oder visualisiert werden.

Die nachfolgende Abbildung zeigt das Inhaltsverzeichnis von JMeter mit den für die Tests verwendeten Konfigurationselementen. Die wichtigsten Elemente sind kurz beschrieben. Der gesamte Aufbau entspricht dem des FOSS4G WMS Benchmarks.

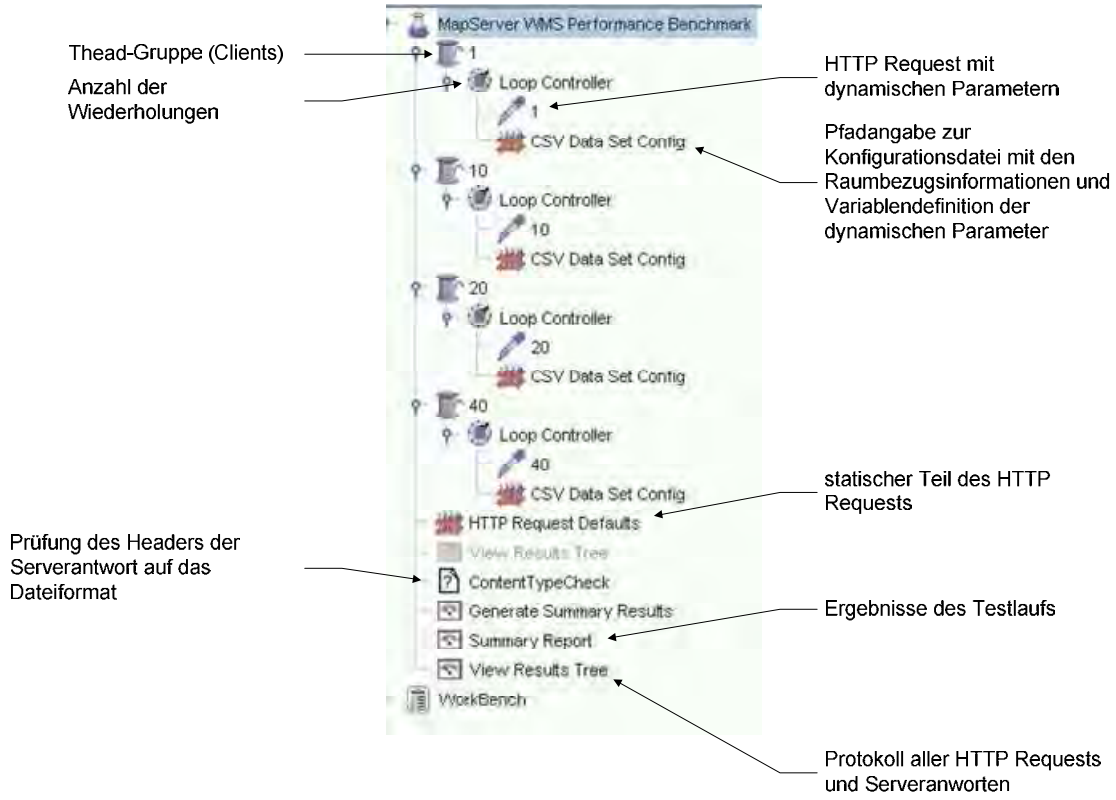


Abb. 29: Aufbau der Konfigurationselemente des Testplans in JMeter (eigener Entwurf)

Folgendes URL-Beispiel zeigt einen Testaufruf:

```
http://192.168.1.3/fcgi-
bin/mapserv.exe?MAP=C:/mapfiles_/wms_rastertest.map&BBOX=4
485006.1,5292651,4487187.3,5294043.6&HEIGHT=588&WIDTH=921&
RE-
QUEST=GetMap&VERSION=1.1.1&SRS=EPSG:31468&FORMAT=image/jpe
g&LAYERS=dop&TRANSPARENT=false
```

Um die vom WMS Server angeforderten Karten überprüfen zu können, wurde der *View Results Tree* Listener verwendet. Folgender Screenshot zeigt einen Listener Datensatz mit der Serverantwort des WMS GetMap Requests:

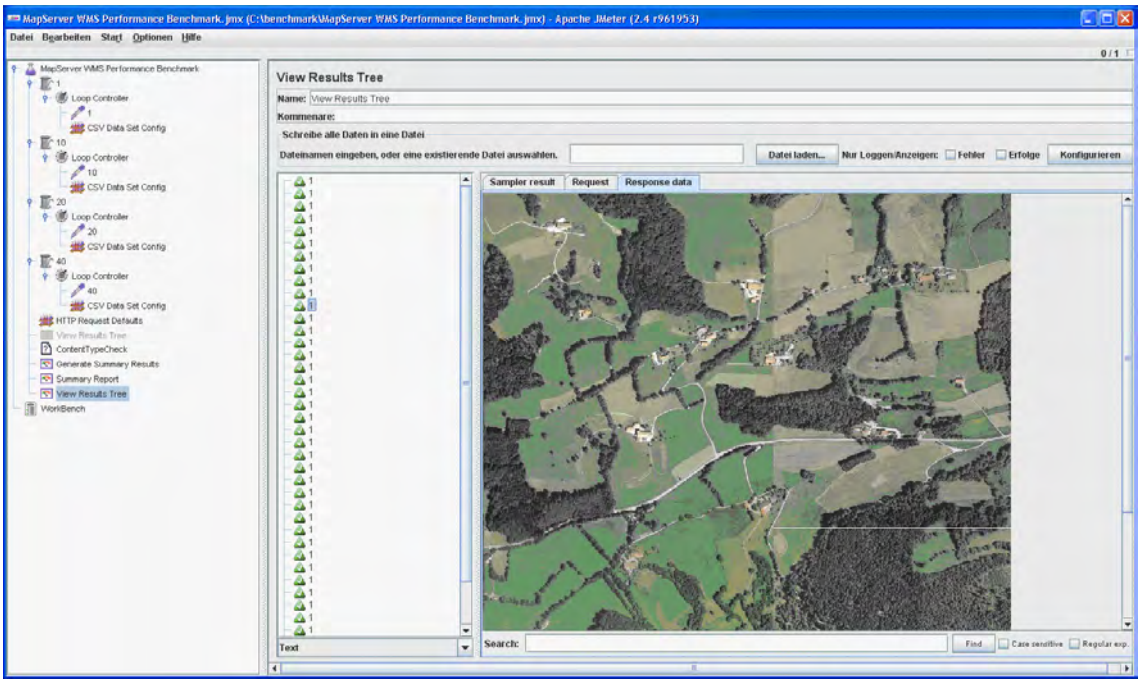


Abb. 30: View Results Tree Listener (eigener Entwurf)

Es wurden für die Tests drei Szenarien entworfen. Im ersten soll ermittelt werden, ob FastCGI oder CGI die performantere Schnittstelle für MapServer UMN ist. Anschließend soll das im Sinne der Performance optimale Datenformat festgestellt werden. In der dritten Testreihe wird der Einfluss des Ausgabeformates auf die Geschwindigkeit geprüft. Alle Szenarien wurden stets inklusive Overviews mit folgendem Ausgabeformat getestet:

```
OUTPUTFORMAT
      NAME jpeg
      DRIVER "AGG/JPEG"
      MIMETYPE "image/jpeg"
      IMAGEMODE RGB
      FORMATOPTION QUALITY=90
END
```

Lediglich beim dritten Szenario wurden mehrere Ausgabeformate verwendet (siehe Anlage 5).

Analog zum FOSS4G MapServer UMN WMS Benchmark ist der Durchsatz die maßgebliche Kennzahl der Tests. Angegeben wird er in Karten pro Sekunde.

5.4 Ergebnisse des Benchmarks

In der ersten Testreihe wurde die MapServer UMN Anwendung mapserv.exe jeweils über die CGI und die FastCGI Schnittstelle angesprochen. Dabei wurden die Original-

formate (TIFF) aller drei Testdatensätze verwendet. Aus diesen insgesamt sechs Testläufen wurde der Mittelwert des Durchsatzes für jede Thread-Gruppe gebildet, um die beiden Schnittstellen miteinander vergleichen zu können.

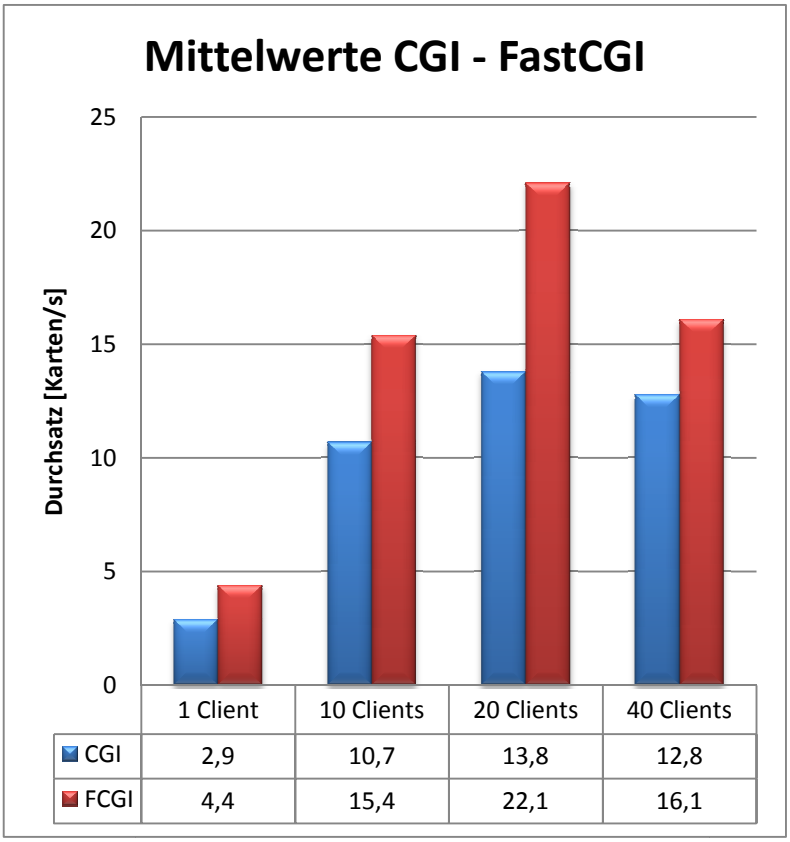


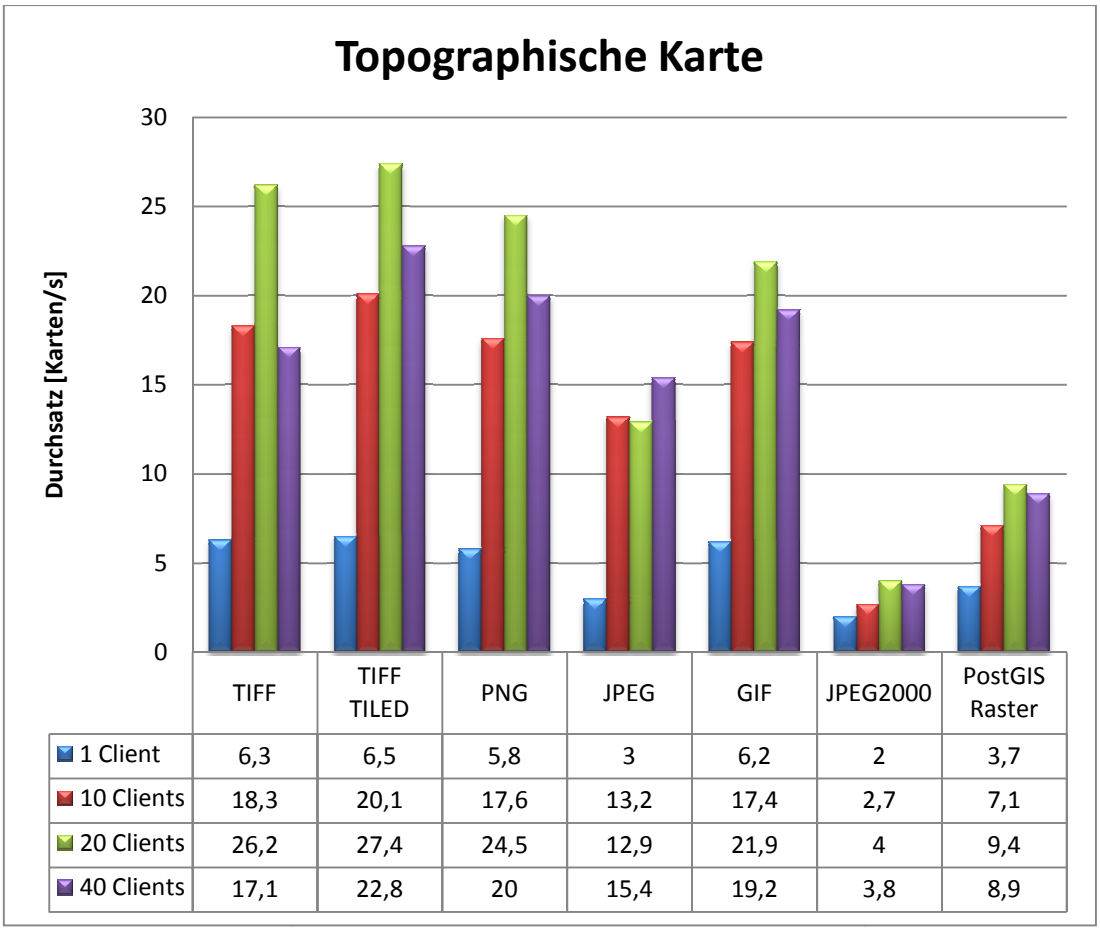
Abb. 31: Testergebnis CGI – FastCGI (eigener Entwurf)

Es zeigt sich ein deutlich höherer Durchsatz bei Verwendung der FastCGI Schnittstelle. Bei 20 Clientzugriffen beträgt der Performancevorteil von FastCGI zu CGI etwa 38 Prozent. Selbst bei der prozentual geringsten Steigerung, bei 40 Clients, liefert MapServer UMN mit FastCGI 3,3 Karten pro Sekunde mehr aus als mit CGI.

Aus diesem Grund wurden alle weiteren Tests nur noch mit FastCGI durchgeführt.

5.4.1 Topographische Karte

Das folgende Diagramm zeigt die Ergebnisse der topographischen Karte. Bis auf JPEG erreicht MapServer UMN mit allen Formaten bei 20 Clientzugriffen seinen maximalen Durchsatz. Optimalstes Format im Hinblick auf die Performance ist das gekachelte TIFF, gefolgt von TIFF und PNG. Bei den restlichen Formaten betragen die Durchsätze teilweise weit unter 20 Karten pro Sekunde.



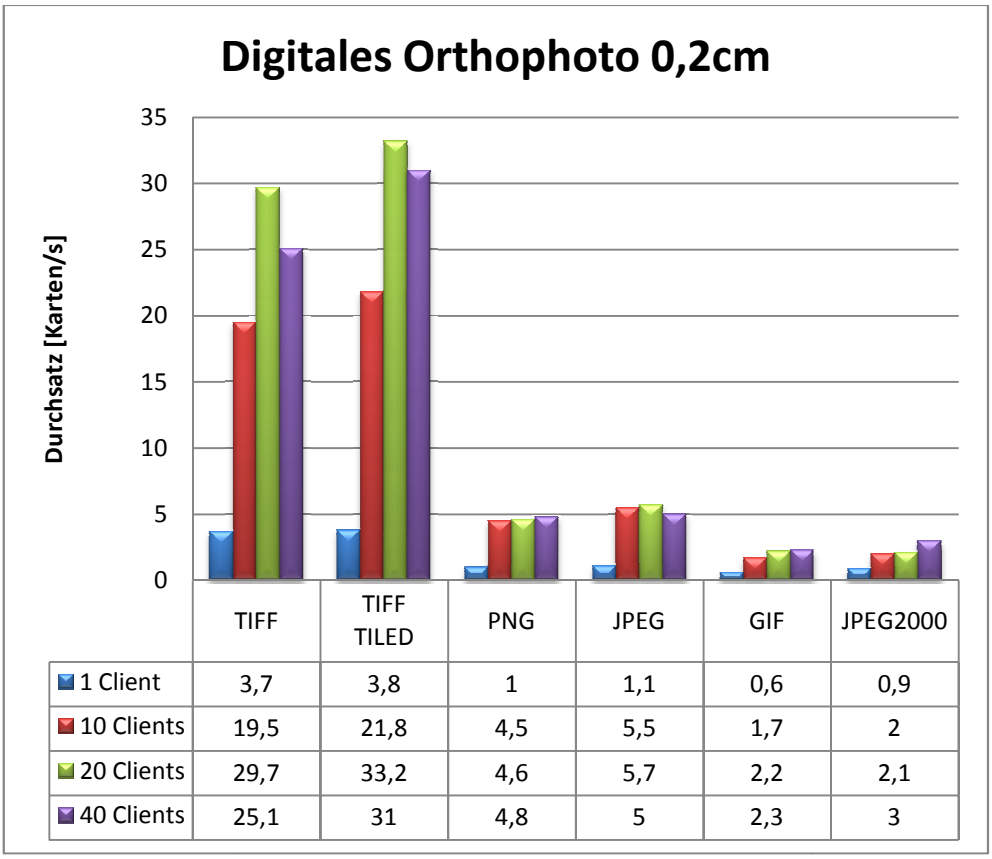
	TIFF	TIFF TILED	PNG	JPEG	GIF	JPEG2000	PostGIS Raster
1 Client	6,3	6,5	5,8	3	6,2	2	3,7
10 Clients	18,3	20,1	17,6	13,2	17,4	2,7	7,1
20 Clients	26,2	27,4	24,5	12,9	21,9	4	9,4
40 Clients	17,1	22,8	20	15,4	19,2	3,8	8,9

Abb. 32: Testergebnis topographische Karte (eigener Entwurf)

Schlusslicht bildet JPEG2000, das mit Abstand die schlechteste Performance bietet. Das Ergebnis von PostGIS Raster liegt signifikant unterhalb der 25 Karten pro Sekunde.

5.4.2 Digitales Orthophoto 0,2cm Bodenauflösung

Die Testreihen mit den Digitalen Orthophotos liefern andere Ergebnisse. Das nachfolgende Diagramm fasst den Durchsatz von MapServer UMN bei Verwendung des hochaufgelösten Orthophotos aus der Straßenbefahrung zusammen.



	TIFF	TIFF TILED	PNG	JPEG	GIF	JPEG2000
1 Client	3,7	3,8	1	1,1	0,6	0,9
10 Clients	19,5	21,8	4,5	5,5	1,7	2
20 Clients	29,7	33,2	4,6	5,7	2,2	2,1
40 Clients	25,1	31	4,8	5	2,3	3

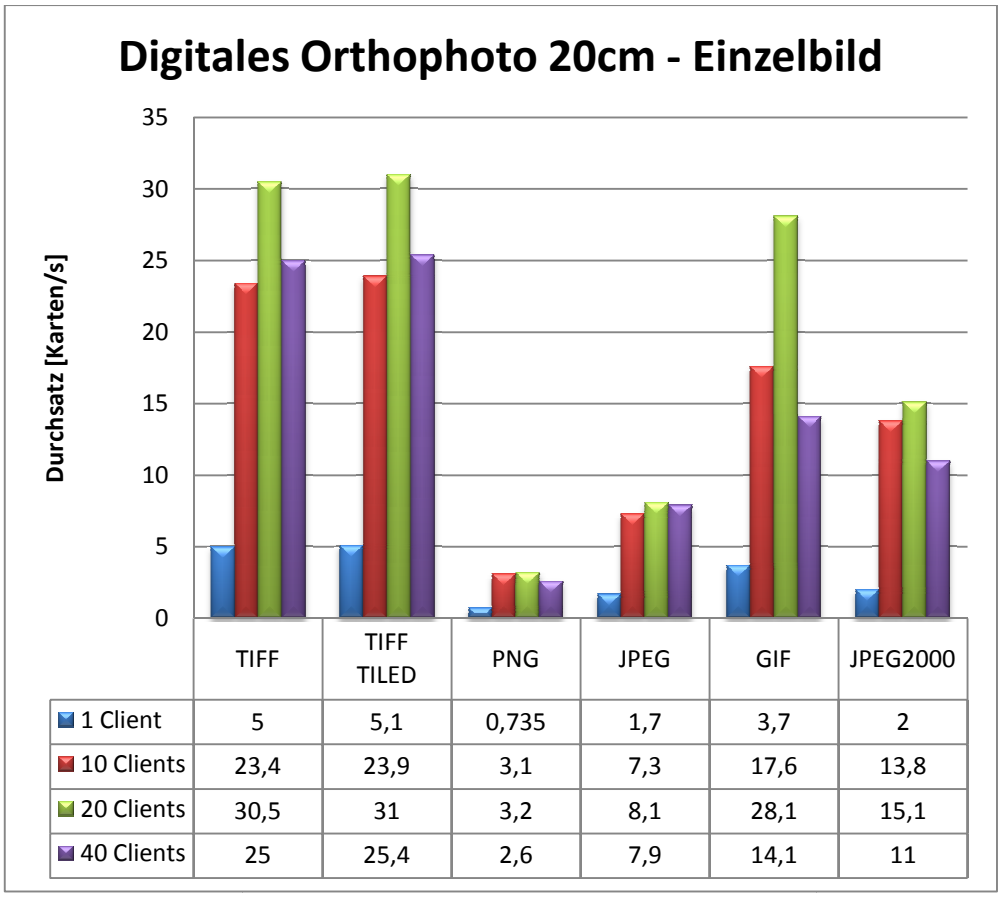
Abb. 33: Testergebnis DOP mit 0,2cm Bodenauflösung (eigener Entwurf)

Mit einem maximalen Durchsatz von 31 Karten pro Sekunde bei 40 Clientzugriffen ist MapServer UMN mit dem gekachelten TIFF am schnellsten. Im TIFF-Format liegt das Ergebnis bei knapp über 25 Karten pro Sekunde bei 40 Clientzugriffen. Mit den anderen vier Formaten ist MapServer UMN extrem „träge“. Mehr als 5,0 Karten pro Sekunde können lediglich mit JPEG erreicht werden. Zu berücksichtigen ist die Tatsache, dass das Ausgangsraster eine Dateigröße von 390 MB besitzt und die Formate mit Kompression weitaus geringere Dateigrößen aufweisen.

5.4.3 Digitales Orthophoto 20cm Bodenauflösung

Die Digitalen Orthophotos mit 20cm Bodenauflösung wurden zweimal getestet. Im ersten Fall wurde nur ein Einzelbild als MapServer UMN Layer dem Benchmark unterzogen. Anschließend wurden die zur Verfügung stehenden 100 Raster mit einem Tileindex zu einem Layer gruppiert und getestet.

Auch beim Einzelbild zeigt sich, dass MapServer UMN mit gekachelten TIFF-Dateien die beste Performance erzielt. Die Ergebnisse von GIF und JPEG2000 unterscheiden sich jedoch von denen des 0,2cm Orthophotos.

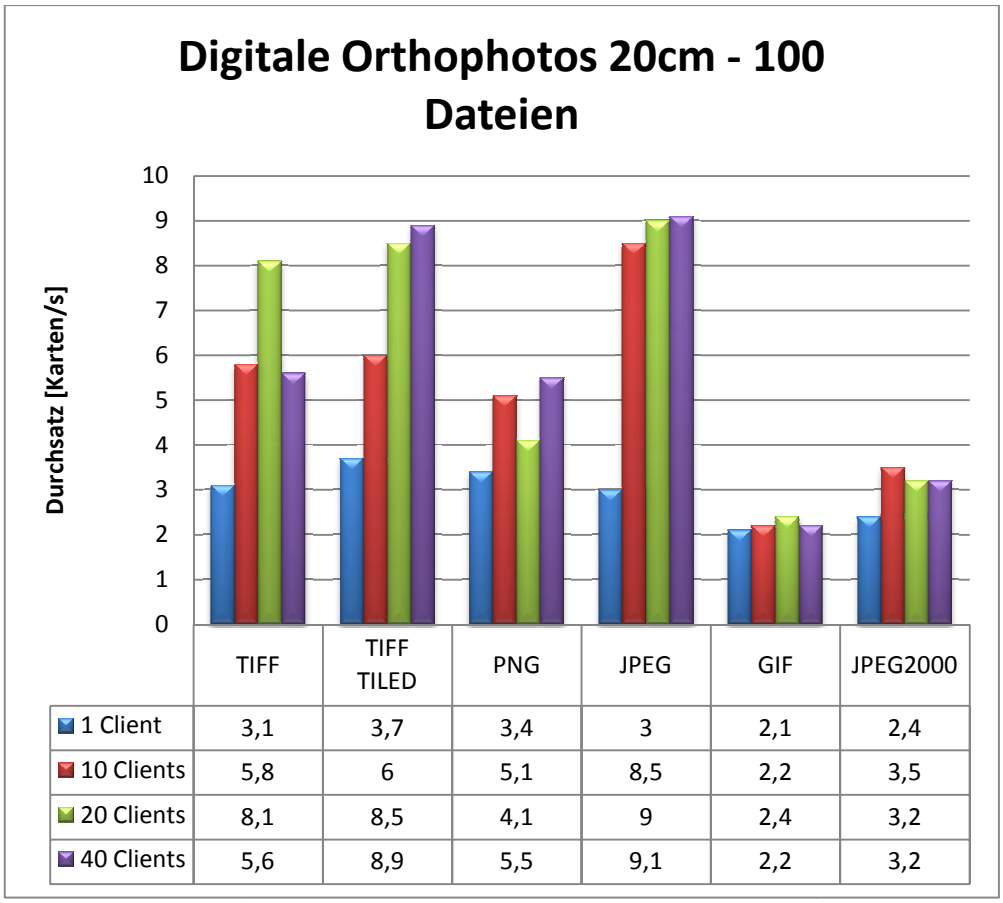


	TIFF	TIFF TILED	PNG	JPEG	GIF	JPEG2000
1 Client	5	5,1	0,735	1,7	3,7	2
10 Clients	23,4	23,9	3,1	7,3	17,6	13,8
20 Clients	30,5	31	3,2	8,1	28,1	15,1
40 Clients	25	25,4	2,6	7,9	14,1	11

Abb. 34: Testergebnis DOP mit 20cm Bodenauflösung - Einzelbild (eigener Entwurf)

Die Zusammenfassung mehrerer Raster zu einem Layer unter Verwendung eines Tileindex ist ein realitätsnahes Szenario. Die Ergebnisse unterscheiden sich sowohl anhand der absoluten als auch der relativen Durchsätze von denen der Einzelbilder. Die Maxima liegen unter 10 Karten pro Sekunde. Ab 20 Clientzugriffen bewirken JPEG Dateien höhere Durchsätze als alle anderen Formate. Der Durchsatz liegt auch bei 40 Clientzugriffen mit 9,1 Karten pro Sekunde minimal über dem des gekachelten TIFF. Mit GIF und JPEG2000 sind die Durchsätze am geringsten.

Es scheint, dass in diesem Fall die JPEG Kompression und die Dateigröße positiven Einfluss auf die Performance haben. Dies ist plausibel, da jetzt gleichzeitig mehrere Raster bzw. Bilder in den Speicher geladen und verarbeitet werden müssen. Dies kann bei einer gewissen Dateigröße mehr Zeit in Anspruch nehmen als das Lesen und Dekomprimieren von kleinen Dateien. Folgendes Diagramm enthält die Ergebnisse des Tests:



	TIFF	TIFF TILED	PNG	JPEG	GIF	JPEG2000
1 Client	3,1	3,7	3,4	3	2,1	2,4
10 Clients	5,8	6	5,1	8,5	2,2	3,5
20 Clients	8,1	8,5	4,1	9	2,4	3,2
40 Clients	5,6	8,9	5,5	9,1	2,2	3,2

Abb. 35: Testergebnis DOP 20cm Bodenauflösung – 100 Bilder (eigener Entwurf)

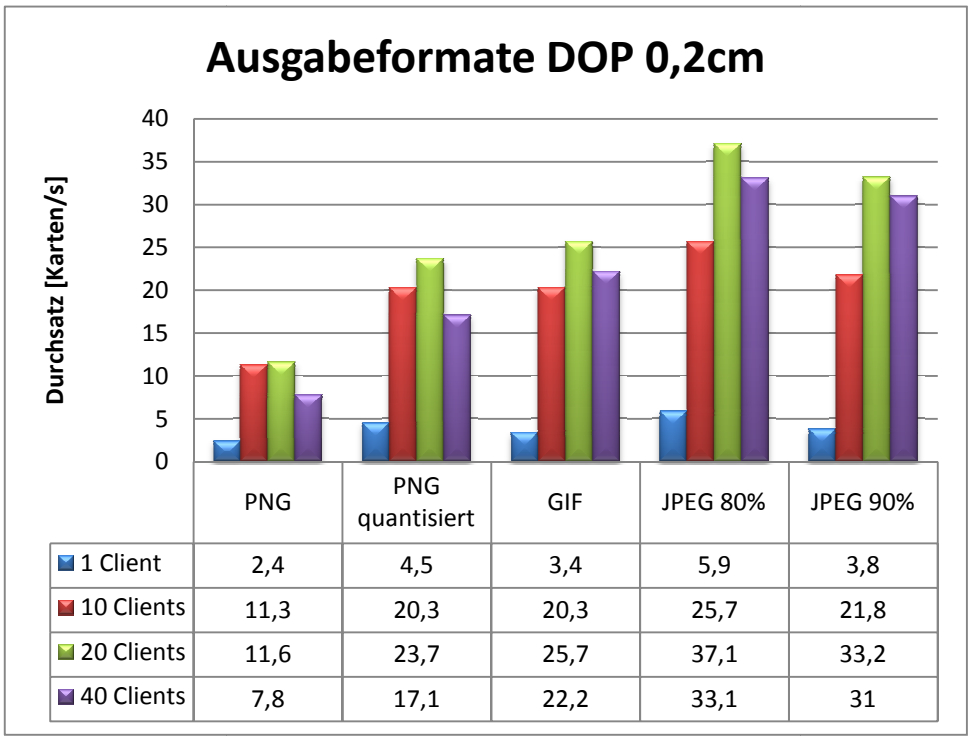
5.4.4 Einfluss des Ausgabeformats

Im letzten Testszenario soll der Einfluss des Ausgabeformates auf die Performance ermittelt werden. Getestet wurden nur Formate, die von gängigen Browsern unterstützt werden. Dies ist in erster Linie PNG, JPEG und GIF. Das BMP-Format wird nicht getestet, da es nicht komprimiert ist (vgl. SAMPLE und IOUP, 2010).

Diese Testreihe wurde mit dem Digitalen Orthophoto (0,2cm Bodenauflösung) und dem einzelnen Digitalen Orthophoto mit 20cm Bodenauflösung durchgeführt.

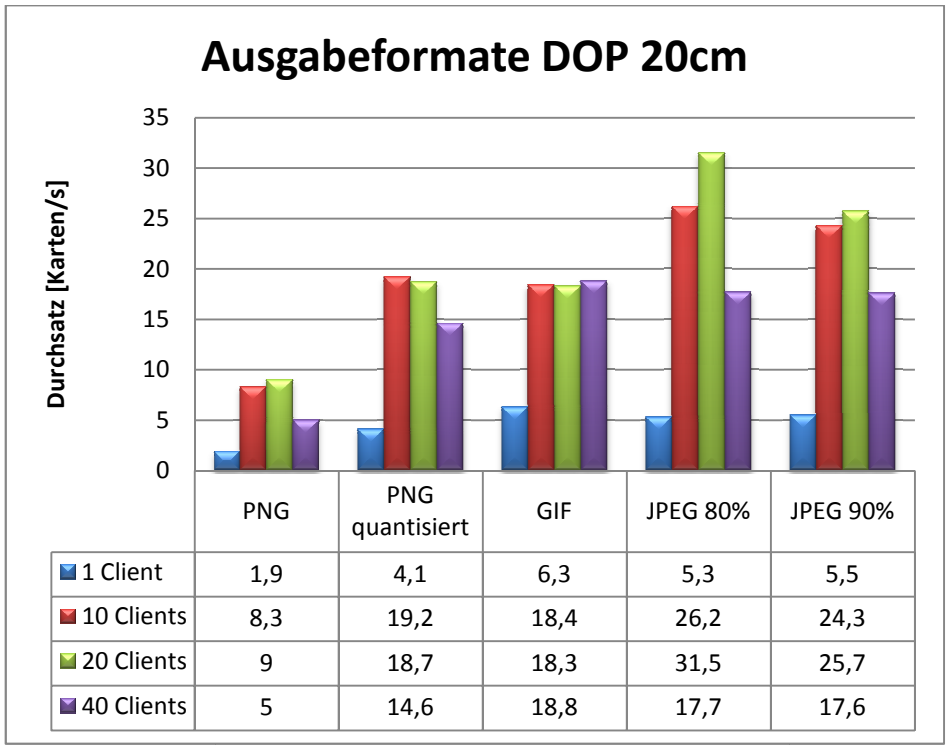
JPEG wurde in 80- und 90-prozentiger Kompression getestet. Geringere Kompressionsraten enthalten erkennbare Kompressionsartefakte. MapServer UMN bietet die Möglichkeit, PNG mit der GD oder AGG Bibliothek in 8-bit Raster zu quantisieren und somit die Dateigröße zu verringern. Dazu muss im Mapfile das FORMATOPTION Schlüsselwort mit der Option “QUANTIZE_FORCE=ON” verwendet werden (vgl. LIME et al., 2011). Die für diesen Test im Mapfile verwendeten OUTPUTFORMAT Angaben sind im Anhang unter Anlage 5 zu finden.

Die Abbildungen zeigen, dass JPEG die größten Durchsätze ermöglicht, gefolgt von GIF und quantisiertem PNG.



	PNG	PNG quantisiert	GIF	JPEG 80%	JPEG 90%
1 Client	2,4	4,5	3,4	5,9	3,8
10 Clients	11,3	20,3	20,3	25,7	21,8
20 Clients	11,6	23,7	25,7	37,1	33,2
40 Clients	7,8	17,1	22,2	33,1	31

Abb. 36: Testergebnis Ausgabeformate DOP 0,2cm (eigener Entwurf)



	PNG	PNG quantisiert	GIF	JPEG 80%	JPEG 90%
1 Client	1,9	4,1	6,3	5,3	5,5
10 Clients	8,3	19,2	18,4	26,2	24,3
20 Clients	9	18,7	18,3	31,5	25,7
40 Clients	5	14,6	18,8	17,7	17,6

Abb. 37: Testergebnis Ausgabeformate DOP 20cm (eigener Entwurf)

5.5 Schlussfolgerung

Es kann anhand der Testergebnisse festgehalten werden, dass das TIFF-Format die besten Ergebnisse im Hinblick auf die Performance liefert, vor allem dann, wenn nur eine Datei verwendet wird. Mit internen Kacheln kann der Durchsatz nochmals erhöht werden. Als Ausgabeformat ist JPEG vorzuziehen.

Allerdings zeigt das Ergebnis der 100 Raster mit Tileindex, dass ab einer gewissen kritischen Masse die Dekompression wesentlich kleinerer Dateien schneller vonstattengeht als das Laden unkomprimierter, großer TIFF-Dateien. Es sind zwar die Ergebnisse von gekacheltem TIFF und JPEG in diesem Fall ähnlich, aber letzteres Dateiformat hat dann einen entscheidenden Vorteil, den es zu berücksichtigen gilt – den geringeren Speicherplatzbedarf.

Das Digitale Orthophoto aus der Straßenbefahrung im TIFF Format hat eine Dateigröße von knapp 390 MB, als JPEG beträgt sie nur circa 16 MB. Die Einsparung von Speicherplatz beträgt somit fast 96 Prozent. Selbst bei den Orthophotos mit 20 cm Bodenauflösung können immerhin knapp 76 Prozent pro Datei eingespart werden. Je nach Kompressionsrate können diese Werte noch gesteigert werden.

Somit scheint die dateibasierte Speicherung von Rasterdaten im JPEG Format auf den ersten Blick wirtschaftlicher zu sein, ohne dass detaillierte Kosten- / Nutzenrechnungen durchgeführt wurden. Nichtsdestotrotz ist zusätzlicher Aufwand nötig, um die Daten zu konvertieren und anschließend die Konvertierungsergebnisse zu überprüfen. Auch der Informationsverlust aufgrund der Kompression muss berücksichtigt werden.

Da für die vorliegende Arbeit Festplattenkapazität keinen limitierenden Faktor darstellt, wurde das TIFF-Format mit internen Kacheln als Datengrundlage für den WMS Dienst verwendet.

6 Aufbau des WebGIS Portals

Nach der Ermittlung der optimalen Konfiguration des WMS Dienstes für die Bereitstellung der Rasterdaten, werden in folgendem Kapitel die Anforderungen analysiert sowie der Aufbau des Portals, dessen Konfiguration und die verwendeten Daten vorgestellt.

6.1 Anforderungen

Vor der Umsetzung des Portals sind dessen Anforderungen zu spezifizieren. Im konkreten Fall sind in einem WebGIS Portal Straßenbefahrungsdaten zu visualisieren. Somit dient es hauptsächlich als Informationssystem für das Infrastrukturvermögen einer Organisation. Dementsprechend muss es übersichtlich und anwenderfreundlich gestaltet sein.

Inhaltlich soll es neben den Basisdaten, wie beispielsweise topographischen Karten und den Netzdaten, auch nachträglich veredelte Daten, wie die Straßenzustandsbewertung, enthalten.

Der Client soll neben den Standardfunktionen (Zoom, Pan und Drucken) auch Bemaßungsfeatures bieten und dem Anwender das Editieren bestimmter Objekte erlauben. Dokumente, die bestimmten Objekten zugeordnet sind, müssen entweder als Download zur Verfügung stehen oder im lokalen Netzwerk direkt angesprochen werden können. Es muss eine Lösung ohne Plug-Ins oder zusätzlicher Software verwendet werden. Ein Standardbrowser muss die Rolle des Clients erfüllen können.

Die Performance des Systems ist enorm wichtig. Daneben soll durch die Verwendung von Standards die Interoperabilität und damit die Nachhaltigkeit des Systems gewährleistet werden.

6.2 Architektur

Das System besteht aus drei physischen oder virtuellen Servern, die gemeinsam die Grundlage des WebGIS Portals bilden. Der mapAccel Applikationsserver stellt dabei den zentralen Knotenpunkt dar. Er stellt mit Hilfe von Apache Tomcat und mapAccel IMS den Client und die Applikationslogik zur Verfügung. Um die Stabilität des ganzen Systems zu erhöhen und die Performance nicht unnötig einzuschränken, werden die Daten getrennt vorgehalten. Vektordaten werden auf einem eigenen Server im Geodatenbankmanagementsystem PostgreSQL mit PostGIS verwaltet. Rasterdaten werden über einen WMS Server bereitgestellt. Auf Grundlage der durchgeführten Benchmarks

wurde sich für eine dateibasierte Datenhaltung im TIFF-Format entschieden. Folgende Abbildung zeigt die Architektur des Systems:

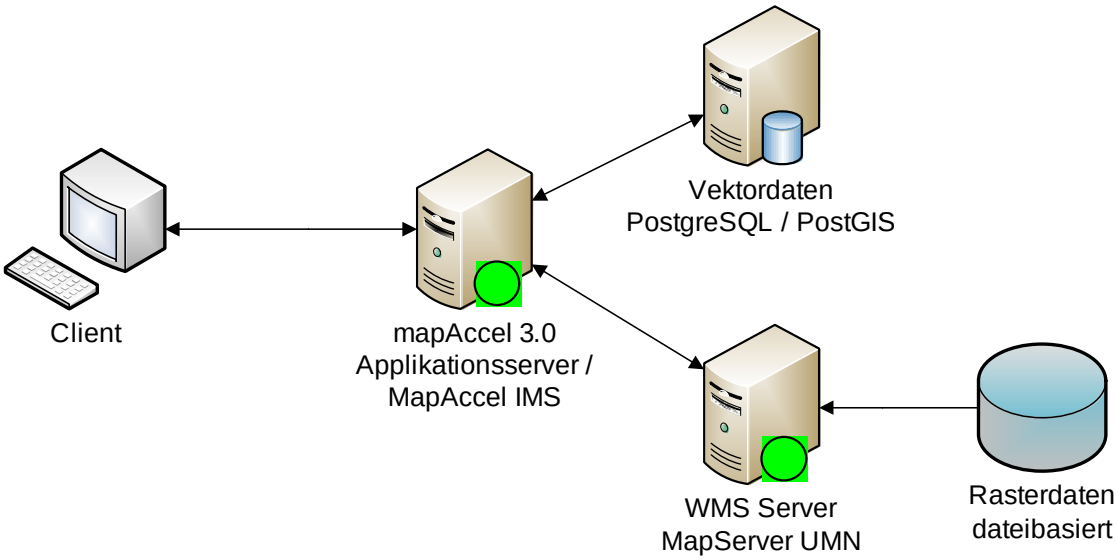


Abb. 38: Architektur des WebGIS Portals (eigener Entwurf)

mapAccel IMS kann dabei selbstverständlich nicht nur als WMS Client fungieren, sondern auch als WMS und WFS Server die Daten weiter zur Verfügung stellen.

Folgende Softwarekonfiguration wurde für die Server verwendet:

MapAccel Applikationsserver	WMS Server	Datenbankserver
• Apache Tomcat 6.0.26 • MapAccel 3.0 • MapAccel IMS 2.4	• Apache 2.2.15 • mod_fcgid 2.3.5 • MapServer UMN 5.6.3 • GDAL 1.7.1	• PostgreSQL 9.0 • PostGIS 1.5.2-3

Abb. 39: Softwarekonfiguration des WebGIS Portals (eigener Entwurf)

6.2.1 Daten

Die aus der Straßenbefahrung erhobenen Daten und deren Aufbereitung werden im folgenden Abschnitt kurz beschrieben. Die zu visualisierenden Daten lassen sich grob in drei Kategorien einteilen:

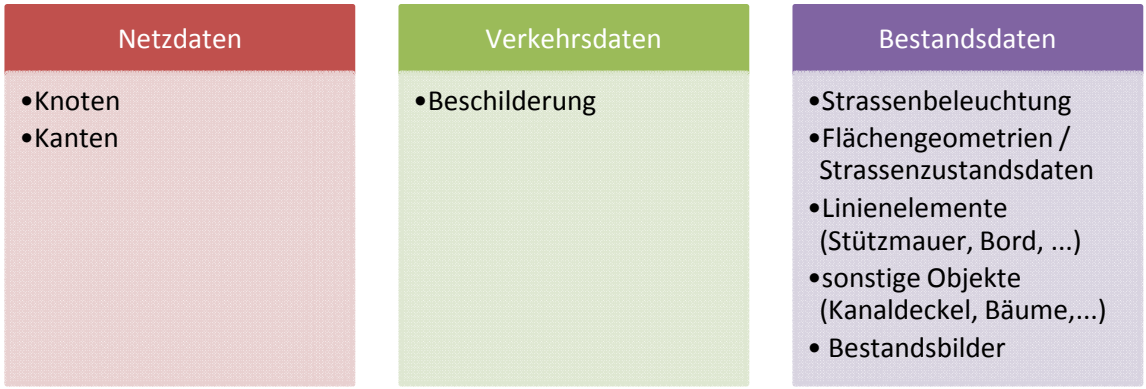


Abb. 40: Übersicht Straßenbefahrungsdaten (eigener Entwurf)

Das Knoten-Kanten-Modell bildet die Grundlage des Straßennetzes. Es liegt als Punkt- und Linien-Shapefile vor und muss für die Visualisierung nicht weiter aufbereitet werden. Es enthält den Straßenschlüssel und die Straßenabschnitte als Attributfelder.

Die Beschilderungsdaten enthalten den Typ des Schildes sowie eine eindeutige ID aus Straßenschlüssel und Abschnitt. Die Schilder sollen anhand der amtlichen Nummern klassifiziert und mit dem jeweiligen Schildsymbol dargestellt werden. Dazu wurden die Zeichen zu den amtlichen Nummern auf Grundlage von THUBAUVILLE (1999) erstellt. Die verwendeten Verkehrszeichen sind im Anhang unter Anlage 6 aufgelistet.

Die Straßenbeleuchtung liegt als Punktthema vor und ist nicht klassifiziert darzustellen.

Ein Polygon-Shapefile enthält die Straßenzustandsdaten. Es besitzt folgende Attribute:

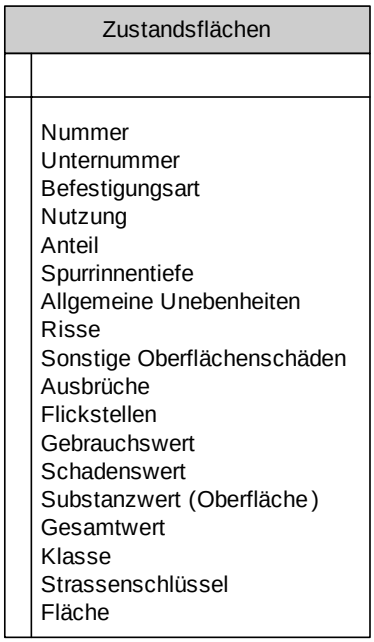


Abb. 41: Attribute der Zustandsflächen (eigener Entwurf)

Die Klassifikation soll anhand des Gesamtwertes erfolgen. Er stellt eine Bewertung des Oberflächenzustandes von Verkehrsflächen dar und ist ordinalskaliert. Die Werte reichen von 1,0 („sehr gut“) bis 5,0 („sehr schlecht“). Der Wert 1,5 kennzeichnet den *Zielwert*, 3,5 den *Warnwert* und 4,5 den *Schwellenwert* (vgl. FGSV, 2003). Die Klassifikation wird wie folgt vorgenommen:

Klasse	RGB Farbcodierung	Farbe
≤ 1,5	0 0 255	
> 1,5 bis 2,5	21 140 9	
> 2,5 bis 3,5	111 244 96	
> 3,5 bis 4,5	255 255 0	
> 4,5	255 0 0	

Tab. 14: Klassifikation des Gesamtwertes (eigener Entwurf)

Bei den sonstigen Objekten handelt es sich um Kanaldeckel, Straßenabläufe und Bäume. Dies gilt jedoch nur für die Testdaten. Möglich sind hier auch andere Objekte, z.B. Armaturen. Die Darstellung erfolgt mit folgenden Symbolen:

Bezeichnung	Symbol
Kanaldeckel	
Strassenablauf	
Baum	

Tab. 15: Klassifikation sonstiger Objekte (eigener Entwurf)

Die Symbole für Straßenabläufe werden im Mapfile aus zwei übereinanderliegenden Symbolobjekten zusammengesetzt. Bäume beruhen auf einer Bilddatei.

Linienelemente, wie Stützmauern, Borde und Rinnen werden einheitlich visualisiert.

Die Bestandsbilder sind anhand des Punkt-Shapefiles *Kamerapositionen* eindeutig referenziert und können somit ohne großen Aufwand integriert werden. Für jedes Objekt in Kamerapositionen existieren pro Kamerasystem zwei Bilder. Der relative Pfad zur Bilddatei ist in Kamerapositionen angegeben und kann für eine Kameraposition wie folgt aussehen:

```
..\20100408\003\2010040800100143_*
```

Das „*“ dient als Wildcard für die einzelnen Kameras. So steht beispielsweise „2L“ für die linke Kamera des 2. Kamerasystems. Um jedes Bild im WebGIS aufrufen zu kön-

nen, bieten sich zwei Umsetzungsvarianten an. Zum einen kann für jede Kamera eine eigene Spalte mit dem entsprechenden Pfad zum Bild angelegt werden oder die Pfade werden in einer Hyperlinkfunktion dynamisch zusammengesetzt. Im vorliegenden Fall entschied man sich für die erste Variante, wobei ein Datenview eingesetzt wurde, um die Originaldaten nicht zu verändern und beim Austausch der Datentabelle die Feldberechnungen nicht erneut ausführen zu müssen. Folgendes SQL SELECT Statement veranschaulicht wie man mit Hilfe der replace-Funktion Teile eines Strings suchen und ersetzen kann:

```
SELECT
 replace(kamerapositionen.photo,'*', '2L') as Pfad_2L
FROM
 kamerapositionen;
```

Alle Vektordaten wurden mit dem Shapefile Import Werkzeug *shp2pgsql* in eine PostgreSQL / PostGIS Datenbank importiert und standardmäßig mit dem GiST-Index versehen. Ein Beispielaufruf von shp2pgsql für einen Import des Shapefiles *knoten* in das Datenbankschema *eagle-eye* der Datenbank *testdb* sieht wie folgt aus:

```
shp2pgsql -s 31468 -d -D -I D:\knoten.shp eagle_eye.knoten
| psql testdb -U postgres
```

6.2.2 Konfiguration des WebGIS-Clients und des -Applikationsservers

Mit dem mapAccel Back-Office *mapSnap* lässt sich der WebGIS Client und der Applikationsserver konfigurieren. Die durchgeführten Schritte werden in diesem Abschnitt erläutert.

In einem ersten Schritt wurde ein Mapfile mit Hilfe von Quantum GIS erzeugt. Mit dem MapServer-Export Plug-In ist es möglich, komplette Mapfiles oder nur die Layer Elemente zu erstellen. Dazu wurden alle Daten in Quantum GIS geladen und entsprechend klassifiziert. Nach Erstellung des Mapfiles wurde dieses mit einem Texteditor nachbearbeitet und die Karte schließlich mit dem Tool *shp2img* überprüft.

Nach Erzeugung des Mapfiles wurde ein neues mapAccel Konfigurationsprojekt mit dem Namen „eagle-eye“ erstellt. Folgende Abbildung zeigt das Dialogfeld:

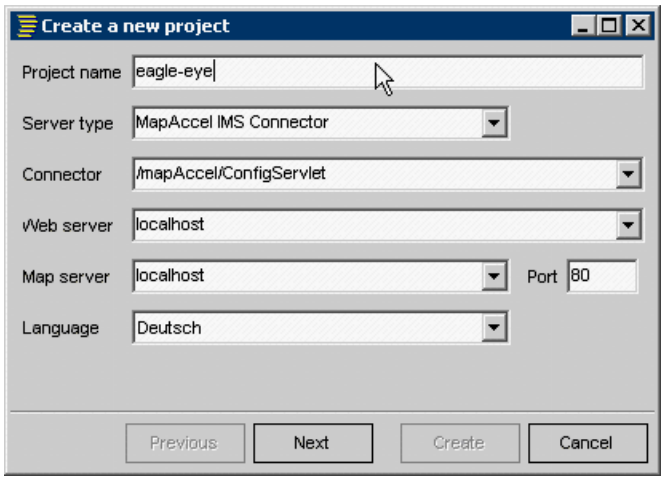


Abb. 42: mapAccel Konfigurationsprojekt anlegen (eigener Entwurf)

Das erzeugte Mapfile dient nun als Grundlage für die mapAccel Konfiguration und muss mit dem Befehl *Load Service Configuration* geladen werden:

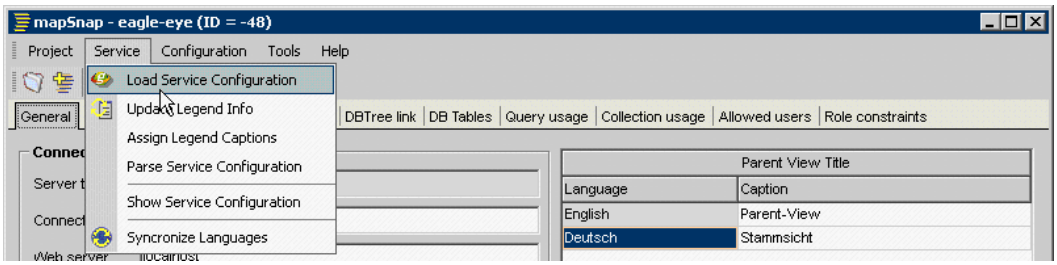


Abb. 43: Mapfile in mapSnap laden (eigener Entwurf)

Nach dem Laden des Mapfiles stehen in mapSnap unter dem Reiter *TOC Configuration* die einzelnen Layer aus dem Mapfile zur Verfügung. Sie sind unter der Gruppe *Group of unused layers* zusammengefasst. Es existieren im Inhaltsverzeichnis von mapSnap *Gruppen*, *Themen* und *Layer*. Gruppen stellen die oberste Ebene dar und können verschiedene Themen enthalten, die wiederum mehrere Layer beinhalten können. Clientseitig sieht der Anwender nur Gruppen und Themen.

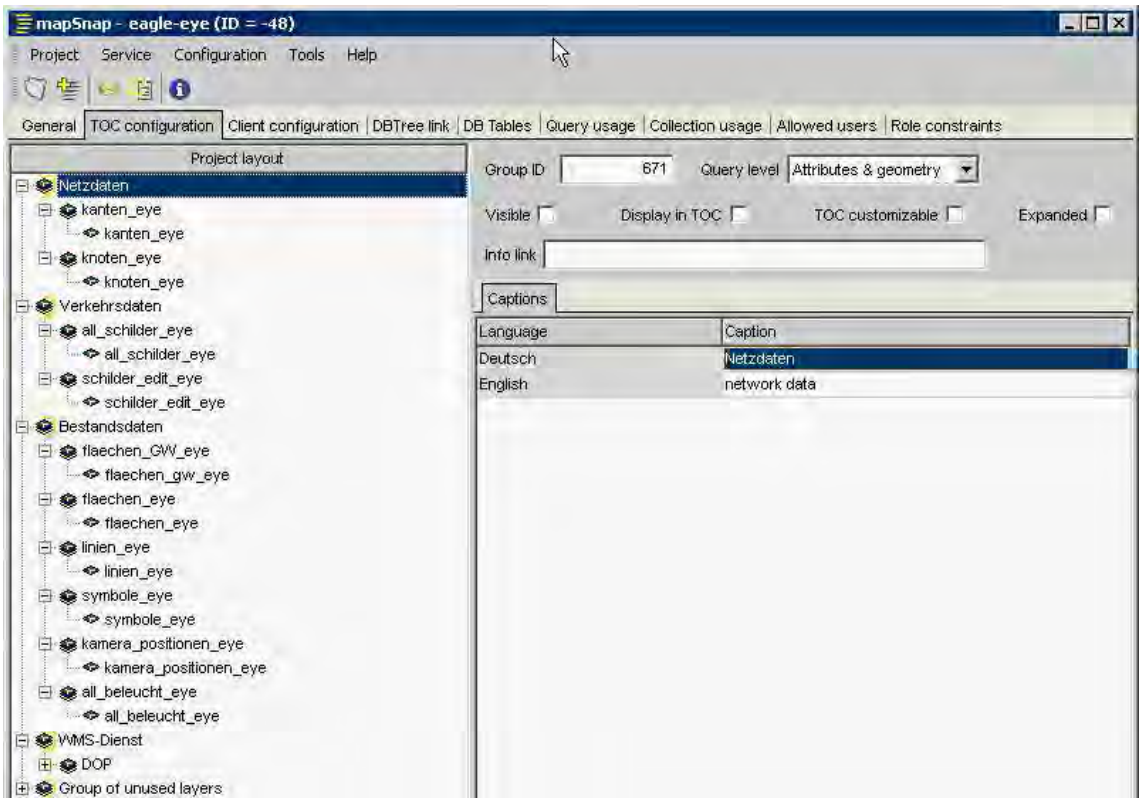


Abb. 44: Gruppen in mapSnap anlegen (eigener Entwurf)

Wie aus der Abbildung ersichtlich ist, wurden vier Gruppen angelegt: Netzdaten, Verkehrsdaten, Bestandsdaten und WMS-Dienst. Die Gruppen wurden jeweils mit Überschriften (Captions) versehen. Auf Themenebene können jene Felder definiert werden, welche im Client sichtbar sind.

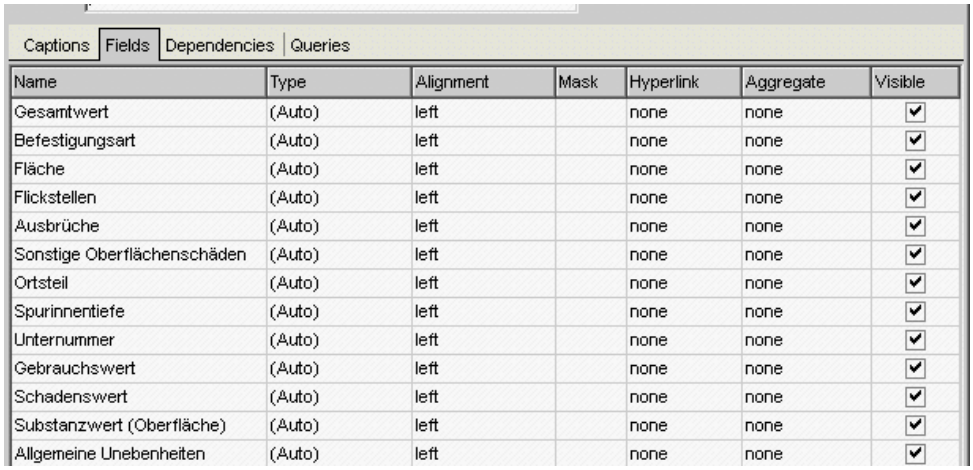


Abb. 45: Felder einrichten in mapSnap (eigener Entwurf)

Neben den in der oberen Abbildung dargestellten Feldern des Layers „Zustandsfläche“, wurden folgende Felder der entsprechenden Layer sichtbar geschaltet:

Layer	**Felder**
Verkehrszeichen	Typ
	Bezeichnung
Verkehrszeichen (editierbar)	Bezeichnung
	erstellt von
	erstellt am
	geändert von
	geändert am
Zustandsfläche (zweiter Layer)	Nutzung
	Befestigungsart
Knoten	Nummer
	Strassenschlüssel
Kanten	Strassenschlüssel
	von Knoten
	bis Knoten
Linienelemente	Nutzung
	Länge
sonstige Objekte	Bezeichnung
Kamerapositionen	Bildnummer
	1 links
	1 rechts
	2 links
	2 rechts
	3 links
	3 rechts

Tab. 16: im Client sichtbare Felder (eigener Entwurf)

Die Felder der Kamerapositionen wurden, bis auf die Bildnummer, als Hyperlink definiert, da über diese die Straßenbefahrungsbilder abgerufen werden sollen. Hyperlink Felder können zum einen so definiert werden, dass die Feldwerte als absolute bzw. relative URLs verwendet werden können. Zum anderen können die Feldwerte an eine JavaScript Funktion übergeben werden. Im vorliegenden Fall wurde die erste Variante bevorzugt, da die Bilder vom Server aus aufgerufen werden sollen. Der relative Dateipfad wird im View zusammengesetzt. Allerdings bietet sich die zweite Möglichkeit an,

wenn auf Bilder im lokalen Dateisystem des Clients mittels ActiveX-Steuerelementen zugegriffen werden soll.

Die Anpassung der Legendenpositionen und -bilder jedes einzelnen Layers kann nachträglich über den Reiter *Legend* erfolgen. Der Reiter enthält alle Informationen aus dem eingelesenen Mapfile. Die Legendenbilder können hier nachträglich ausgetauscht werden. Die Klassen können einzeln aktiviert werden. Folgende Abbildung zeigt die Legendeneinstellung des Layers Zustandsfläche:

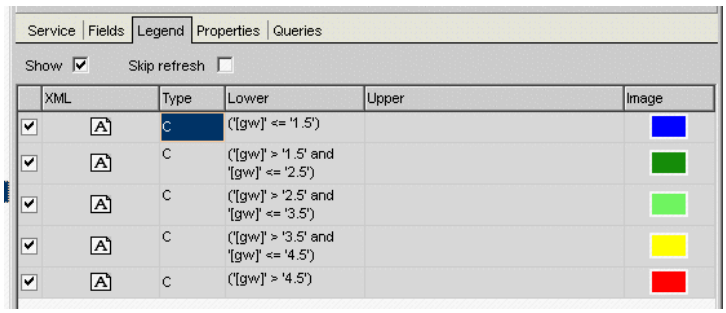


Abb. 46: Legendenbearbeitung in mapSnap (eigener Entwurf)

Suchen, in mapAccel *Queries* genannt, wurden sowohl für den Layer „Zustandsfläche“ als auch für „Verkehrszeichen“ eingerichtet. Hierzu wurde auf Gruppen- bzw. Layerebene über den Reiter *Queries* eine Suchdefinition (*Query Definition*) festgelegt. Mit dieser werden die zu verwendenden Suchfelder und Operatoren konfiguriert. Aktiviert werden die Suchen über den Reiter *Query Usage*.

Das Editieren von Layern wird über den Bereich *Remote editing* des Reiters *Properties* gesteuert. Man kann das Modifizieren, Erzeugen und Löschen von Features bzw. Geometrien einzeln freischalten. Anzugeben ist die Datenbankverbindung und die Datenbanktabelle mit Schema.

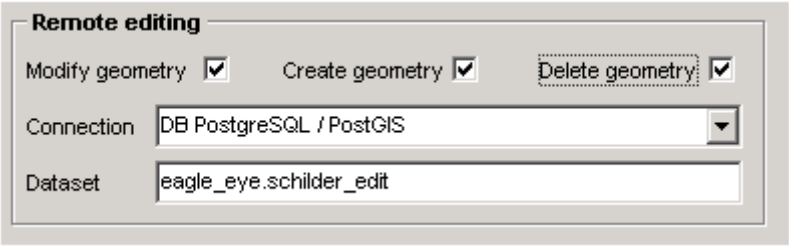


Abb. 47: Editing Einstellungen in mapSnap (eigener Entwurf)

Es können sowohl direkte als auch globale Datenbankverbindungen definiert werden. Letztere haben den Vorteil, dass der Applikationsserver die Verbindung verwaltet und nicht die Webapplikation selbst. Dies kann die Performance steigern.

Hierzu wurde in der Serverkonfigurationsdatei *server.xml* von Apache Tomcat eine *DataSource* im <GlobalNamingResources> Tag definiert. Benötigt werden die Parameter, um sich auf die Datenbank verbinden zu können. Dazu zählt die URL sowie Username und Passwort:

```
<Resource
   name="jdbc/postgis"
   auth="Container"
   driverClassName="org.postgis.jts.JtsGisWrapper"
   url="jdbc:postgresql://192.168.1.2:5432/testDB"
   type="javax.sql.DataSource"
   username="postgres" password="xxxxxxxx" maxActive="20"
   maxIdle="10" maxWait="1"/>
```

In diesem Stadium ist die Datenquelle nur dem Applikationsserver, aber noch nicht der Webapplikation zur Verfügung gestellt. Dies muss wiederum in *server.xml* konfiguriert werden. Es wurde ein *Context* Element innerhalb des <Host> Tags hinzugefügt:

```
<Context docBase="eagle-eye" path="/eagle-eye">
   <ResourceLink
      type="javax.sql.DataSource"
      name="jdbc/postgis"
      global="jdbc/postgis" />
</Context>
```

Bevor die Datenquelle verwendet werden kann, muss Tomcat noch der entsprechende Datenbanktreiber zur Verfügung gestellt werden. Dazu wurde der benötigte PostgreSQL JDBC-Treiber (postgresql-8.4-701.jdbc4.jar) ins *lib* Verzeichnis von Tomcat kopiert.

In mapSnap kann nun die Datenbankverbindung in der Verbindungsverwaltung unter dem Menü *Tools* hinzugefügt werden.

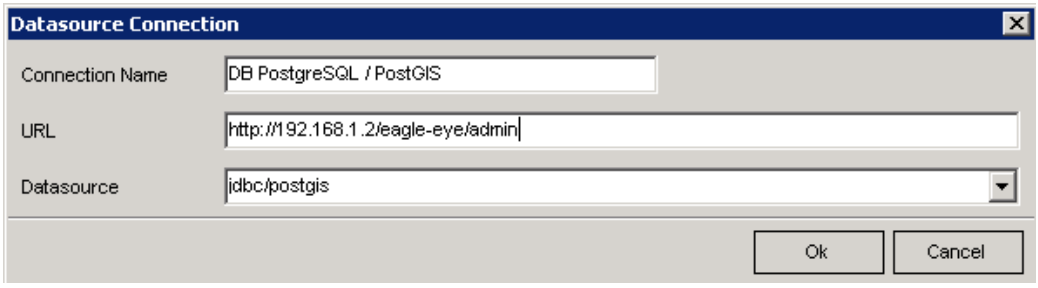


Abb. 48: Datenbankverbindung in mapSnap (eigener Entwurf)

Damit ist die Konfiguration der Editing Einstellungen abgeschlossen. Editierbar konfiguriert wurden die Layer Verkehrszeichen und Zustandsflächen.

Des Weiteren besteht die Möglichkeit, Benutzerrechte zu vergeben und benutzerbezogene, auf der Grundkonfiguration basierende Sichten, einzurichten. Letztere werden als *Child Views* bezeichnet. Für die vorliegende Fragestellung wurden keine eigenen Sichten eingerichtet. Dies kann in größeren Organisationen allerdings von Vorteil sein, wenn nur bestimmte Daten einem eingeschränkten Benutzerkreis zur Verfügung gestellt werden sollen.

Das generelle Verhalten des WebGIS Clients wird über den Reiter *Client Configuration* gesteuert. Hier wurden die räumliche Startausdehnung und die maximale räumliche Ausdehnung des Clients festgelegt. Die Farbeinstellungen der Select-, Highlight- und Buffer-Funktionen sowie die Position und Einheiten der Maßstabsleiste werden unter diesem Reiter verwaltet.

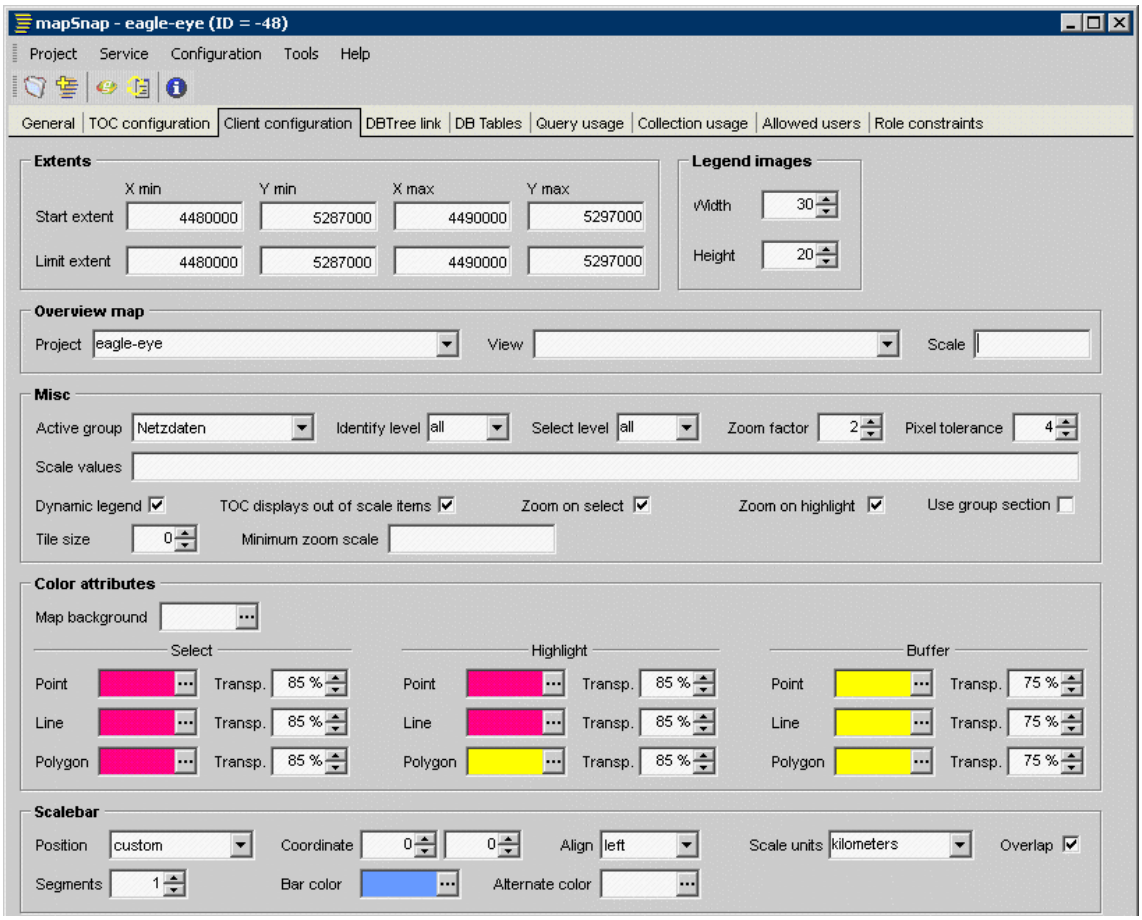


Abb. 49: Reiter Client Configuration (eigener Entwurf)

Als letzter Schritt wurde im Menü *Configuration* der Befehl *Create Config File* ausgeführt. Damit wird die Konfiguration als XML-Datei gespeichert und dem Applikationsserver zur Verfügung gestellt. Mit diesen Schritten ist die Einrichtung des WebGIS Portals beendet.

Folgende Abbildung zeigt den WebGIS Client mit aktivierten Zustandsflächen, Verkehrszeichen, sonstigen Symbolen, dem Knoten-Kanten-Modell und den Kamerapositionen:

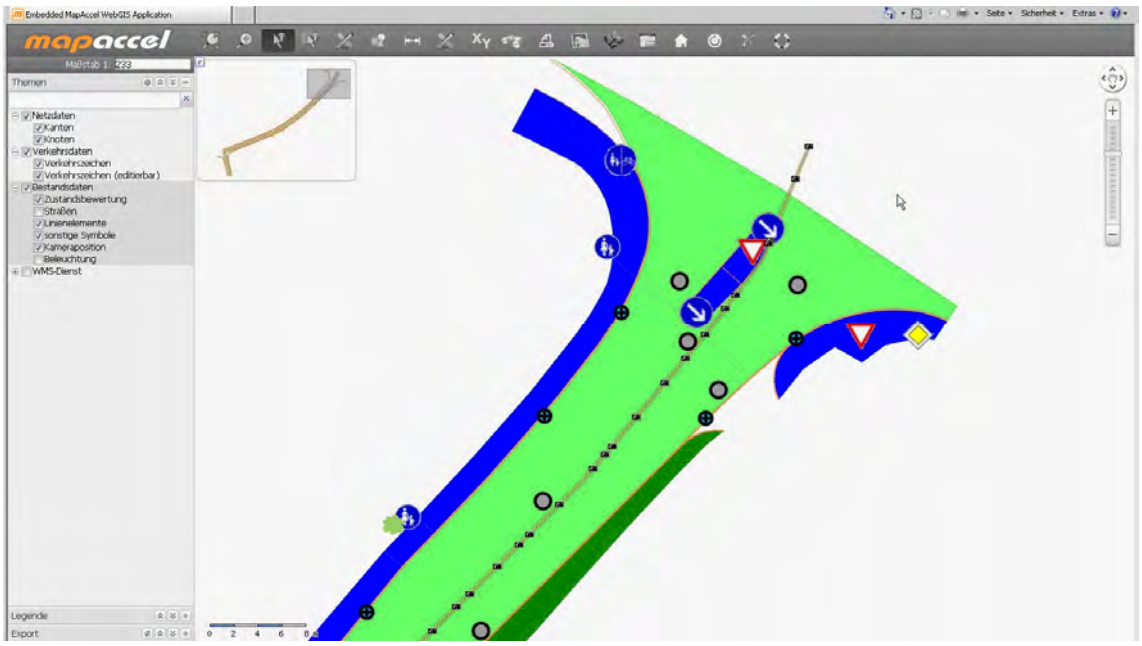


Abb. 50: WebGIS Client mit Straßenbefahrungsdaten (eigener Entwurf)

Weitere Abbildungen sind dem Anhang unter Anlage 8 beigefügt.

7 Abschließende Betrachtungen

7.1 Zusammenfassung

Mit der vorliegenden Arbeit sollten Straßenbefahrungsdaten in einem WebGIS visualisiert werden. Ziel der Arbeit war, einen Kartendienst so zu konfigurieren, dass dieser Rasterdaten mit optimaler Performance zur Verfügung stellen kann. Damit verbunden war die Wahl des Datenformates für die zu verarbeitenden Rasterdaten.

Nach einem kurzen Überblick über den aktuellen Forschungsstand, wurden die theoretischen Grundlagen onlinebasierter Geoinformationstechnologien dargelegt. Im Besonderen wurden dabei standardisierte Dienste vorgestellt. In einem nächsten Schritt wurde der Aufbau einer WebGIS Infrastruktur mit ihren Komponenten skizziert. Die für diesen Zweck optimale Rasterdatenhaltung und Konfiguration wurde mittels eines WMS Benchmarks ermittelt.

Zusammenfassen lassen sich die Ergebnisse des Benchmarks wie folgt:

- Bei Verwendung von FastCGI ist eine höhere Performance zu erzielen.
- Eine dateibasierte Haltung der Rasterdaten ist vorzuziehen.
- MapServer UMN liefert bei Verwendung des TIFF-Formats mit internen Kacheln (tiled TIFF) als Datenquelle den größten Durchsatz.
- Ab einer gewissen Anzahl von Quellrastern ist das JPEG-Format wirtschaftlicher als TIFF und übertrifft dessen Performance.
- Als MapServer UMN Ausgabeformat ist JPEG zu bevorzugen.

Auf Grundlage dieser Erkenntnisse wurde die Konfiguration von MapServer UMN durchgeführt. Als Framework für das zu erstellende WebGIS Portal wurde mapAccel 3.0 von Territorium Online GmbH herangezogen. Dessen Konfiguration wurde anhand der aus der Straßenbefahrung stammenden Daten durchgeführt und erläutert.

7.2 Diskussion

Unterzieht man die Testergebnisse einer kritischen Betrachtung, so müssen Vergleichsdaten herangezogen werden. Dafür eignen sich die Veröffentlichungen von AIME und MCKENNA (2009), die das FOSS4G WMS Benchmark beschreiben und ZETTEL (2007),

der die Performance von MapServer UMN unter Verwendung verschiedener Vektordatenformate überprüft hat.

Erstere haben die Performanceunterschiede von MapServer UMN und GeoServer untersucht. Dabei wurden Rasterdaten getestet, deren Dateigröße als enorm hoch einzustufen ist. Es wurde das bekannte Satellitenbild *Blue Marble Next Generation* der NASA verwendet. Das Raster wurde als Tileindex (512 je 16 GB große TIFF-Dateien) sowie im BigTIFF Format und als ECW getestet. BigTIFF und ECW wurden als Einzelbild in das Mapfile eingebunden (vgl. AIME und MCKENNA, 2009). Insofern können die Ergebnisse von AIME und MCKENNA nur begrenzt mit denen der vorliegenden Arbeit verglichen werden, obwohl die Methodik identisch ist. Unabhängig davon, ist die allgemeine Systemperformance an die eingesetzte Hard- und Software gebunden. Jedoch zeigen auch hier die Ergebnisse, dass MapServer UMN mit FastCGI eine wesentlich höhere Performance aufweist als mit CGI. Ergebnis ihres Rasterdaten Benchmarks zeigt folgende Abbildung:

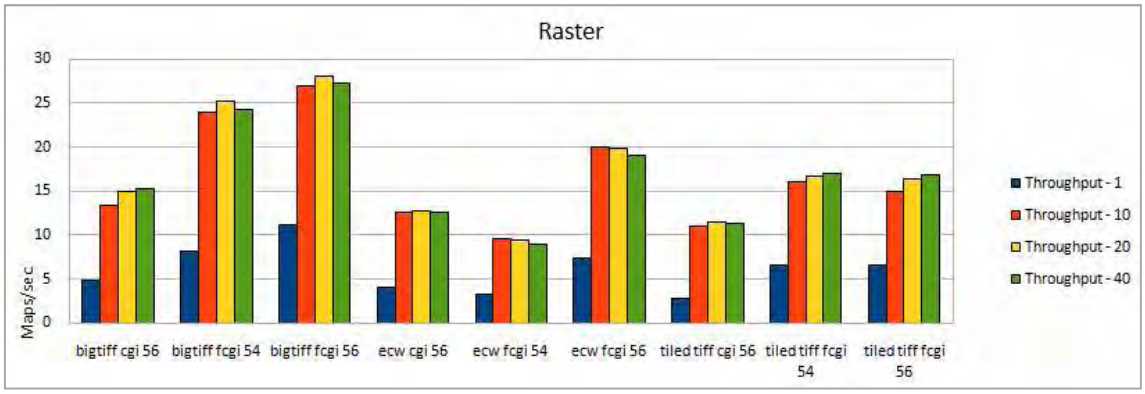


Abb. 51: FOSS4G WMS Benchmark 2009 Ergebnis (AIME und MCKENNA, 2009)

BigTIFF liefert hier die besten Ergebnisse gefolgt von ECW und den TIFFs mit Tileindex. Grundsätzlich zeigt MapServer UMN in der Rasterdatenverarbeitung laut AIME und MCKENNA (2009) eine wesentlich höhere Performance als GeoServer.

ZETTEL (2007) hat in seiner Arbeit unter anderem die MapServer UMN Ausgabeformate GIF, PNG und JPEG getestet und die Antwortzeit in Millisekunden als Kennzahl für die Performance verwendet. Als Datenquelle wurden Vektordaten mit einer abgestuften Anzahl an Datensätzen ins Mapfile eingebunden. Er kommt in seinen Tests zu dem Ergebnis, dass das GIF Ausgabeformat die besten Ergebnisse in Bezug auf die Performance liefert, gefolgt von PNG und JPEG (vgl. ZETTEL, 2007).

Das in der vorliegenden Arbeit von ZETTEL (2007) abweichende Ergebnis kann durch die Eigenschaften der zu erzeugenden Kartenbilder erklärt werden. Aufgrund der verwendeten Rasterdaten, besitzt das von MapServer UMN gerenderte Kartenbild erheblich mehr Informationen in Form von Pixelwerten. Dies hat unterschiedliche Dateigrößen zur Folge. Außerdem verwendet Zettel (2007) in seinen Tests die GD Bibliothek zur Erzeugung der PNG Bilder und nutzt die IMAGEMODE Einstellung PC256 für GIF und PNG. Damit erzeugt MapServer UMN Bilder mit 256 Farben, die zwar eine geringe Dateigröße aufweisen, aber zur Visualisierung von Orthophotos mit 24-bit Farbtiefe völlig ungeeignet sind. Somit ist das Ergebnis der vorliegenden Arbeit plausibel.

Weil die Ergebnisse hard- und softwareabhängig sind, hätten sowohl verschiedenste Betriebssysteme als auch unterschiedliche Kartenserver untersucht werden können. Dies hätte jedoch den Rahmen der Arbeit deutlich überzogen, könnte allerdings die Grundlage neuer Untersuchungen bilden.

Die Arbeit beschränkt sich auf die gängigsten Dateiformate. Zusätzliche Tests mit kommerziellen Rasterdatenformaten, wie beispielsweise *MrSID* (Multiresolution Seamless Image Database) von LizardTech, oder Geodatenbanksystemen, wie *Oracle Spatial*, fehlen.

Für die Visualisierung der Straßenbefahrungsdaten hätten selbstverständlich auch andere WebGIS Applikationen eingesetzt werden können. Jedoch wurde bewusst mapAccel 3.0 verwendet, da es sich um ein neues, innovatives System handelt, über das bisher keine wissenschaftlichen Publikationen vorliegen.

Die gewählte Architektur und die Konfiguration des Clients richten sich nach den Anforderungen, die im Normalfall an ein WebGIS Portal gestellt werden. Sicherheitsaspekte, wie z.B. Authentifizierung, Benutzergruppen oder Sichten, wurden vernachlässigt. Weitere standardisierte Dienste, wie WFS oder WCS, hätten in das System integriert werden können. Da mapAccel IMS auf MapServer UMN basiert, können alle Vorteile im Hinblick auf Interoperabilität, die MapServer UMN bietet, umgesetzt werden. Auf die Beschreibung der Standard WebGIS Funktionalitäten und der Werkzeuge des WebGIS Clients wurde verzichtet, da dies nicht in Zusammenhang mit der Fragestellung steht.

7.3 Ausblick

Der neue Web Map Tile Service Standard des OGC lässt erahnen, wohin die Richtung und Weiterentwicklung hoch performanter WebGIS Anwendungen geht.

Die steigende Popularität der Smart Mapping Clients liegt vor allem in deren Performance beim Bildaufbau begründet, die den schnellen Wechsel in verschiedene Zoomstufen und das beliebige Verschieben des Kartenausschnitts ermöglichen. Das Schlagwort für die Technik, die dafür verantwortlich ist, lautet *Tiling*. Dabei werden die Originaldaten in kleinere gekachelte Bilder zerlegt. Für verschiedene diskrete Zoomstufen, die an einen fixen Maßstab gebunden sind, werden die Kacheln erstellt und auf einem Server abgelegt. Wird von einem Client ein bestimmter Kartenausschnitt in einem bestimmten Maßstab angefordert, so werden ihm nur diejenigen Kacheln zur Verfügung gestellt, die das betreffende Gebiet in dem gewünschten Maßstab abdecken. Da die einzelnen Kacheln bereits existieren, müssen sie nicht erst vom Kartendienst gerendert werden. Diese Anfragen belasten den Server deshalb nur minimal (vgl. SAMPLE und IOUP, 2010). Dieser Umstand macht das Tiling vor allem für statische Daten, die sich nicht oft ändern, wie beispielsweise Orthophotos oder topographische Karten, interessant.

Bei der Tile-basierten Speicherung von Rasterdaten sind neben der Wahl der eigentlichen Tilegröße auch die in dieser Arbeit bereits vorgestellten PNG- und JPEG-Dateiformate zu berücksichtigen. Der Speicherplatzbedarf kann unter Umständen enorm hoch sein. Wenn man sich vereinfachend quadratische Tiles vorstellt, so beträgt für jede diskrete Zoomstufe *i*, die Anzahl der Tiles $2^i \cdot 2^{i-1}$ (vgl. SAMPLE und IOUP, 2010). Der durch das Tiling erreichte Performancegewinn rechtfertigt oftmals diesen hohen Speicherbedarf.

Es existieren bereits verschiedene Tileserver, die Tiles erzeugen und zur Verfügung stellen können. *MapProxy* (http://mapproxy.org/), ein OpenSource Produkt, zählt dazu. MapProxy kann auch andere WMS Server cachen und wiederum als WMS Server fungieren. Damit lassen sich vor allem öffentliche, oft langsame WMS Dienste erheblich beschleunigen.

Die Untersuchung dieser Tileserver und der Tile-basierten Speicherung von Daten kann neue Erkenntnisse liefern und bisherige Ergebnisse sinvoll ergänzen.

Das WebGIS Portal besitzt neben der reinen Visualisierung der Befahrungsdaten zweifelsohne das Potenzial, zu einem Pavement-Management-System ausgebaut zu werden.

8 Literaturverzeichnis

BARTELME, N. (2005): Geoinformatik. Modelle Strukturen Funktionen, 4. vollständig überarbeitete Auflage, Berlin, Heidelberg

BAUMANN, P. (2009): Array Databases and Raster Data Management. In: ÖZSU, T., LIU, L. [Hrsg.] (2009): Encyclopedia of Database Systems

BÄUMKER, M. und J. LUDWIG (2007): „eagle eyes technology"- ein kinematisches terrestrisches photogrammetrisches Stereoaufnahmesystem mit direkter Georeferenzierung mittels INS, GPS und Odometer. In: CHESI/WEINOLD, Hrsg.: 14. Internationale Geodätische Woche Obergurgl 2007, S. 1–10, Obergurgl, Österreich

BRINKHOFF, T. (2008): Geodatenbanksysteme in Theorie und Praxis. Einführung in objektrelationale Geodatenbanken unter besonderer Berücksichtigung von Oracle Spatial. 2., überarbeitete und erweiterte Auflage. Heidelberg

BURGER, W., BURGE, M. J. (2009): Principals of Digital Image Processing. Fundamental Techniques. London

CHEN, R., XIE, J. (2008): Open Source Databases and Their Spatial Extensions. In: HALL, G.B., LEAHY, M.G. [Hrsg.] (2008): Open Source Approaches in Spatial Data Handling. Advances in Geographic Information Science 2. Heidelberg

FISCHER, S. (2002): Grafikformate GE-PACKT. Bonn

FORSCHUNGSGESELLSCHAFT FÜR STRAßEN- UND VERKEHRSWESEN – Arbeitsgruppe: Sonderaufgaben, Arbeitsausschuss: Systematik der Straßenerhaltung, Arbeitskreis: Erhaltung kommunaler Straßen (2003): Empfehlungen für das Erhaltungsmanagement von Innerortsstraßen, E EMI 2003. Ausgabe 2003. Köln

HARING, A. (2007): Die Orientierung von Laserscanner- und Bilddaten bei der fahrzeuggestützten Objekterfassung. Dissertation am Institut für Photogrammetrie und Fernerkundung der Technischen Universität Wien

JANSEN, M., ADAMS, T. (2010): OpenLayers. Webentwicklung mit dynamischen Karten und Geodaten. München

KORDUAN, P. und ZEHNER, M. L. (2008): Geoinformation im Internet. Technologien zur Nutzung raumbezogener Informationen im WWW. Heidelberg

KRALIDIS, A.T. (2008): Geospatial Open Source and Open Standards Convergences. In: HALL, G.B., LEAHY, M.G. [Hrsg.] (2008): Open Source Approaches in Spatial Data Handling. Advances in Geographic Information Science 2. Heidelberg

KROPLA, B. (2005): Beginning MapServer: Open Source GIS Development. New York

LANDRATSAMT MIESBACH (2011): Screenshot des mapAccel WebGIS Clients. Miesbach

LIME, S. (2008): MapServer. In: HALL, G.B., LEAHY, M.G. [Hrsg.] (2008): Open Source Approaches in Spatial Data Handling. Advances in Geographic Information Science 2. Heidelberg

MCIHAGGA, D. (2008): Communities of Practice and Business of Open Source Web Mapping. In: HALL, G.B., LEAHY, M.G. [Hrsg.] (2008): Open Source Approaches in Spatial Data Handling. Advances in Geographic Information Science 2. Heidelberg

MITCHELL, T. (2008). Web-Mapping mit Open Source-GIS-Tools. Überarbeitet von EMDE, A. und CHRISTL, A., Köln

MURRAY, J. D. und VANRYPER, W. (1996): Encyclopedia of Graphics File Formats, Second Edition. Sebastopol

PENG, Z.-R., TSOU, M.-H. (2003): INTERNET GIS. Distributed Geographic Information Services for the Internet and Wireless Networks. Hoboken

SAMPLE, J. T., IOUP, E. (2010): Tile-Based Geospatial Information Systems. Principles and Practices. New York

SIMON, D., SCHAAB, G., DÜPMEIER, C. (2004): Intelligente Schnittstellen zur Integration von Kartendiensten in Web-basierte Informationssysteme am Beispiel des „Themenpark Boden, Geologie und Natur“. Diplomarbeit am Institut für Angewandte Informatik der Fachhochschule Karlsruhe

STRUTZ, T. (2009): Bilddatenkompression. Grundlagen, Codierung, Wavelets, JPEG, MPEG, H.264, 4. Auflage. Heidelberg

TERRITORIUM ONLINE GMBH (2010a): mapAccel. Das GIS Applikations Framework für Enterprise Systeme. Application-Server und WebClient Handbuch. Bozen

TERRITORIUM ONLINE GMBH (2010b): mapAccel. Das GIS Applikations Framework für Enterprise Systeme. mapAccel Backoffice Handbuch „mapSnap“. Bozen

THUBAUVILLE, W. (1999): Straßenverkehrs-Ordnung (StVO) und Allgemeine Verwaltungsvorschrift zur StVO (VwV-StVO) mit Verkehrszeichenkatalog (VzKat). Textausgabe mit Einführung. Köln

WALLS, C. (2009): Modular Java. Creating Flexible Applications with OSGi and Spring. Raleigh

WARMERDAM, F. (2008): The Geospatial Data Abstraction Library. . In: HALL, G.B., LEAHY, M.G. [Hrsg.] (2008): Open Source Approaches in Spatial Data Handling. Advances in Geographic Information Science 2. Heidelberg

Internetquellen

AIME, A. und MCKENNA, J. (2009): WMS Performance Shootout: http://svn.osgeo.org/osgeo/foss4g/benchmarking/scripts/results/2009/benchmarking2009.odp, letzter Zugriff am 19.03.2011

ANTI-GRAIN GEOMETRY, http://www.antigrain.com/, letzter Zugriff am 20.03.2011

APACHE JMETER, http://jakarta.apache.org/jmeter/, letzter Zugriff am 01.03.2011

APACHE TOMCAT: http://tomcat.apache.org/, letzter Zugriff am 20.03.2011

BAYERISCHES STAATSMINISTERIUM FÜR UMWELT UND GESUNDHEIT: Naturschutzgebiete Bayerns, Web Map Service der Umweltverwaltung: http://s-stmugv0072.umwelt.bayern.de/arcwms/com.esri.wms.Esrimap?request=getcapabilities&service=wms, letzter Zugriff am 14.02.2011

EAGLE EYE TECHNOLOGIES GmbH: http://www.ee-t.de, letzter Zugriff am 13.01.2011

FOSS4G BENCHMARK: http://wiki.osgeo.org/wiki/FOSS4G_Benchmark

GARNETT, J. J. (2005): Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/ideas/essays/archives/000385.php, letzter Zugriff am 03.02.2011

GDAL OGR UTILITY PROGRAMS, http://www.gdal.org/ogr_utilities.html, letzter Zugriff am 20.03.2011

GDAL UTILITIES, http://www.gdal.org/gdal_utilities.html, letzter Zugriff am 20.03.2011

GDAL VIRTUAL FORMAT TUTORIAL: http://www.gdal.org/gdal_vrttut.html, letzter Zugriff am 10.02.2011

GDALADDO: http://www.gdal.org/gdaladdo.html, letzter Zugriff am 20.02.2011

GiST for PostgreSQL: http://www.sai.msu.su/~megera/postgres/gist/, letzter Zugriff am 25.02.2011

HostGIS (2007): Tile Indexes. In: The Map Server Team (2011): MapServer Documentation. Release 5.6.6. http://mapserver.org/MapServer.pdf

HostGIS (2008): Raster. In: The Map Server Team (2011): MapServer Documentation. Release 5.6.6. Veröffentlicht online unter http://mapserver.org/MapServer.pdf

PostGIS Windows experimental binaries, http://www.postgis.org/download/windows/experimental.php, letzter Zugriff am 20.02.2011

IrfanView, http://www.irfanview.de/, letzter Zugriff am 14.03.2011

Li, L., Li, J., Yu T. (2002): The Study on Web GIS Architecture based on JNLP. Symposium on Geospatial Theory, Processing and Applications, Ottawa. http://www.isprs.org/proceedings/XXXIV/part4/pdfpapers/271.pdf

libdg, http://www.libgd.org, letzter Zugriff am 03.02.2011

Lime, S., McKenna, J., Doyon J.F. (2011): Mapfile. In: The Map Server Team (2011): MapServer Documentation. Release 5.6.6. http://mapserver.org/MapServer.pdf

MapProxy, http://mapproxy.org/, letzter Zugriff am 19.03.2011

MapServer for Windows – MS4W, http://www.maptools.org/ms4w/index.phtml, letzter Zugriff am 20.03.2011 OSGeo4W (http://trac.osgeo.org/osgeo4w/)

MapServer UMN Data Input, http://mapserver.org/input/index.html, letzter Zugriff am 20.03.2011

MapServer UMN FAQ, http://mapserver.org/faq.html, letzter Zugriff am 20.03.2011

MapServer UMN Mapfile Referenz, http://mapserver.org/mapfile/index.html, letzter Zugriff am 20.03.2011

MapServer UMN, http://mapserver.org, letzter Zugriff am 20.03.2011

McKenna, J. (2010a): WMS Client. In: The Map Server Team (2011): MapServer Documentation. Release 5.6.6. http://mapserver.org/MapServer.pdf, letzter Zugriff am 20.03.2011

McKenna, J. (2010b): WMS Server. In: The Map Server Team (2011): MapServer Documentation. Release 5.6.6. http://mapserver.org/MapServer.pdf, letzter Zugriff am 20.03.2011

NACIONALES, P. S. (2009): AntiAliasing with MapServer. In: The Map Server Team (2011): MapServer Documentation. Release 5.6.6. http://mapserver.org/MapServer.pdf, letzter Zugriff am 03.03.2011

NACIONALES, P. S., MCKENNA, J. (2010): MapServer Tutorial. http://mapserver.org/tutorial/index.html, letzter Zugriff am 15.02.2011

OGC (2005): Web Feature Service Implementation Specification. Version 1.1.0, http://www.opengeospatial.org/standards/wfs, letzter Zugriff am 05.03.2011

OGC (2006): OpenGIS ® Web Map Server Implementation Specification. Version 1.3.0, http://www.opengeospatial.org/standards/wms, letzter Zugriff am 20.03.2011

OGC (2007): Styled Layer Descriptor profile of the Web Map Service Implementation Specification. Version 1.1.0, http://www.opengeospatial.org/standards/sld, letzter Zugriff am 01.03.2011

OGC (2010): OpenGIS® Web Map Tile Service Implementation Standard, http://www.opengeospatial.org/standards/wmts, letzter Zugriff am 14.03.2011

OGC Standards and Specifications: http://www.opengeospatial.org/standards, letzter Zugriff am 20.03.2011

OPEN MARKET, INC. (1996): FastCGI: A High-Performance Web Server Interface. Technical White Paper: http://www.fastcgi.com/drupal/node/6?q=node/15, letzter Zugriff am 03.03.2011

OPENLAYERS EXAMPLES, http://openlayers.org/dev/examples/, letzter Zugriff am 22.02.2011

OPENLAYERS MAP VIEWER SERVICE, http://trac.osgeo.org/openlayers/wiki/MapViewerService, letzter Zugriff am 22.02.2011

OPENSOURCE GEOSPATIAL FOUNDATION TILE MAP SERVICE SPECIFICATION: http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification, letzter Zugriff am 30.01.2011

OpenSource GEOSPATIAL FOUNDATION: http://www.osgeo.org, letzter Zugriff am 16.03.2011

OSGeo4W, http://trac.osgeo.org/osgeo4w/, letzter Zugriff am 20.03.2011

PostGIS Raster Documentation: http://trac.osgeo.org/postgis/wiki/WKTRaster/Documentation01, letzter Zugriff am 19.02.2011

PostGIS Raster: http://trac.osgeo.org/postgis/wiki/WKTRaster, letzter Zugriff am 20.02.2011

PostGIS Ticket #497, http://trac.osgeo.org/postgis/ticket/497, letzter Zugriff am 20.03.2011

Promberger, K., Früh, G., Niederkofler, R. (2004): Neues kommunales Haushalts- und Rechnungswesen in der Bundesrepublik Deutschland. Working Paper 14/2004. http://www.verwaltungsmanagement.at/622/uploads/working_paper_14.pdf, letzter Zugriff am 16.02.2011

Quantum GIS, http://qgis.org/, letzter Zugriff am 14.03.2011

Ramsey, P. (2007): The State of Open Source GIS. http://2007.foss4g.org/presentations/viewattachment.php?attachment_id=8

Statistisches Bundesamt (2010a): Verkehrsmittelbestand und Infrastruktur, Verkehrsinfrastruktur in Deutschland (1 000 km), http://www.destatis.de/jetspeed/portal/cms/Sites/destatis/Internet/DE/Content/Statistiken/Verkehr/VerkehrsmittelbestandInfrastruktur/Tabellen/Content75/Verkehrsinfrastruktur,templateId=renderPrint.psml

Statistisches Bundesamt (2010b): Verkehr im Überblick - Stand 08.10.2010 - Fachserie 8 Reihe 1.2 – 2009, https://www-ec.destatis.de/csp/shop/sfg/bpm.html.cms.cBroker.cls?cmspath=struktur,vollanzeige.csp&ID=1026148, letzter Zugriff am 15.03.2011

Tyagi, S. (2006): RESTful Web Services. http://www.oracle.com/technetwork/articles/javase/index-137171.html, letzter Zugriff am 22.02.2011

Warmerdam, F. (2009): wms_request.py: http://svn.osgeo.org/osgeo/foss4g/benchmarking/scripts/wms_request.py, letzter Zugriff am 01.03.2011

9 Anhang

Anlage 1: Von GDAL 1.7.0 unterstützte Rasterdatenformate (gdalinfo –formats):

rw = Lese- und Schreibzugriff

ro = Lesezugriff

v = Virtual I/O

VRT (rw+): Virtual Raster
GTiff (rw+v): GeoTIFF
NITF (rw+v): National Imagery Transmission Format
RPFTOC (ro): Raster Product Format TOC format
HFA (rw+v): Erdas Imagine Images (.img)
SAR_CEOS (ro): CEOS SAR Image
CEOS (ro): CEOS Image
JAXAPALSAR (ro): JAXA PALSAR Product Reader (Level 1.1/1.5)
GFF (rov): Ground-based SAR Applications Testbed File Format (.gff)
ELAS (rw+): ELAS
AIG (ro): Arc/Info Binary Grid
AAIGrid (rwv): Arc/Info ASCII Grid
SDTS (ro): SDTS Raster
OGDI (ro): OGDI Bridge
DTED (rwv): DTED Elevation Raster
PNG (rwv): Portable Network Graphics
JPEG (rwv): JPEG JFIF
MEM (rw+): In Memory Raster
JDEM (ro): Japanese DEM (.mem)
GIF (rwv): Graphics Interchange Format (.gif)
BIGGIF (rov): Graphics Interchange Format (.gif)
ESAT (ro): Envisat Image Format
BSB (ro): Maptech BSB Nautical Charts
XPM (rw): X11 PixMap Format
BMP (rw+v): MS Windows Device Independent Bitmap
DIMAP (ro): SPOT DIMAP
AirSAR (ro): AirSAR Polarimetric Image
RS2 (ro): RadarSat 2 XML Product
PCIDSK (rw+v): PCIDSK Database File
PCRaster (rw): PCRaster Raster File
ILWIS (rw+v): ILWIS Raster Map
SGI (rw+): SGI Image File Format 1.0
SRTMHGT (rwv): SRTMHGT File Format
Leveller (rw+): Leveller heightfield
Terragen (rw+): Terragen heightfield

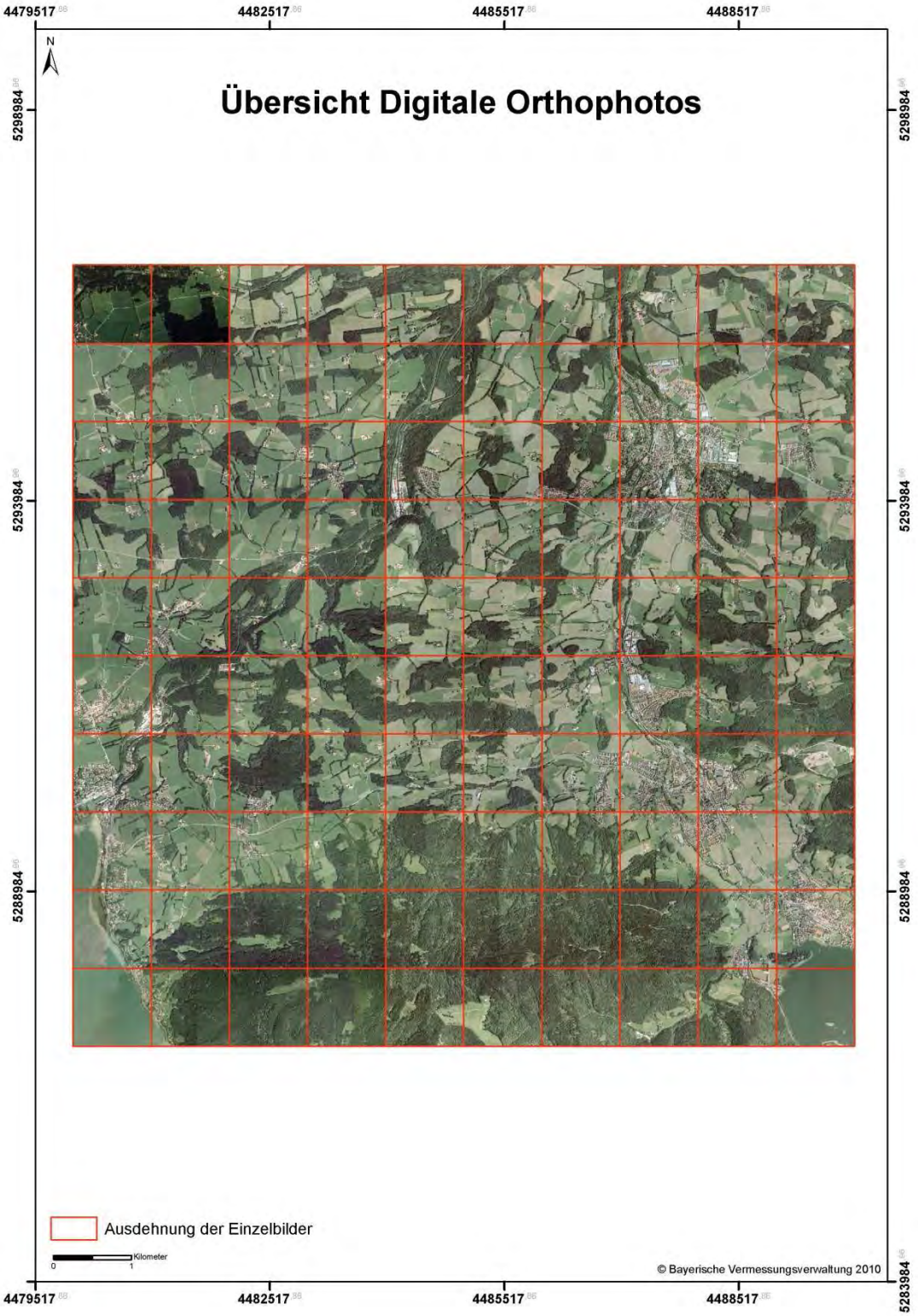
GMT (rw): GMT NetCDF Grid Format
netCDF (rw): Network Common Data Format
HDF4 (ro): Hierarchical Data Format Release 4
HDF4Image (rw+): HDF4 Dataset
ISIS3 (ro): USGS Astrogeology ISIS cube (Version 3)
ISIS2 (ro): USGS Astrogeology ISIS cube (Version 2)
PDS (ro): NASA Planetary Data System
TIL (ro): EarthWatch .TIL
ERS (rw+): ERMapper .ers Labelled
ECW (rw): ERMapper Compressed Wavelets
JP2ECW (rw+): ERMapper JPEG2000
L1B (ro): NOAA Polar Orbiter Level 1b Data Set
FIT (rw): FIT Image
GRIB (ro): GRIdded Binary (.grb)
MrSID (ro): Multi-resolution Seamless Image Database (MrSID)
JP2MrSID (ro): MrSID JPEG2000
RMF (rw+): Raster Matrix Format
WCS (ro): OGC Web Coverage Service
WMS (ro): OGC Web Map Service
MSGN (ro): EUMETSAT Archive native (.nat)
RST (rw+): Idrisi Raster A.1
INGR (rw+): Intergraph Raster
GSAG (rw): Golden Software ASCII Grid (.grd)
GSBG (rw+): Golden Software Binary Grid (.grd)
GS7BG (ro): Golden Software 7 Binary Grid (.grd)
COSAR (ro): COSAR Annotated Binary Matrix (TerraSAR-X)
TSX (ro): TerraSAR-X Product
COASP (ro): DRDC COASP SAR Processor Raster
R (rwv): R Object Data Store
PNM (rw+): Portable Pixmap Format (netpbm)
DOQ1 (ro): USGS DOQ (Old Style)
DOQ2 (ro): USGS DOQ (New Style)
ENVI (rw+v): ENVI .hdr Labelled
EHdr (rw+v): ESRI .hdr Labelled
GenBin (ro): Generic Binary (.hdr Labelled)
PAux (rw+): PCI .aux Labelled
MFF (rw+): Vexcel MFF Raster
MFF2 (rw+): Vexcel MFF2 (HKV) Raster
FujiBAS (ro): Fuji BAS Scanner Image
GSC (ro): GSC Geogrid
FAST (ro): EOSAT FAST Format
BT (rw+): VTP .bt (Binary Terrain) 1.3 Format
LAN (ro): Erdas .LAN/.GIS
CPG (ro): Convair PolGASP

IDA (rw+): Image Data and Analysis
NDF (ro): NLAPS Data Format
EIR (ro): Erdas Imagine Raw
DIPEx (ro): DIPEx
LCP (rov): FARSITE v.4 Landscape File (.lcp)
RIK (ro): Swedish Grid RIK (.rik)
USGSDEM (rw): USGS Optional ASCII DEM (and CDED)
GXF (ro): GeoSoft Grid Exchange Format
HTTP (ro): HTTP Fetching Wrapper
BAG (ro): Bathymetry Attributed Grid
HDF5 (ro): Hierarchical Data Format Release 5
HDF5Image (ro): HDF5 Dataset
NWT_GRD (ro): Northwood Numeric Grid Format .grd/.tab
NWT_GRC (ro): Northwood Classified Grid Format .grc/.tab
ADRG (rw+v): ARC Digitized Raster Graphics
SRP (rov): Standard Raster Product (ASRP/USRP)
BLX (rw): Magellan topo (.blx)
Rasterlite (rw): Rasterlite
WKTRaster (ro): PostGIS WKT Raster driver
SAGA (rw+v): SAGA GIS Binary Grid (.sdat)

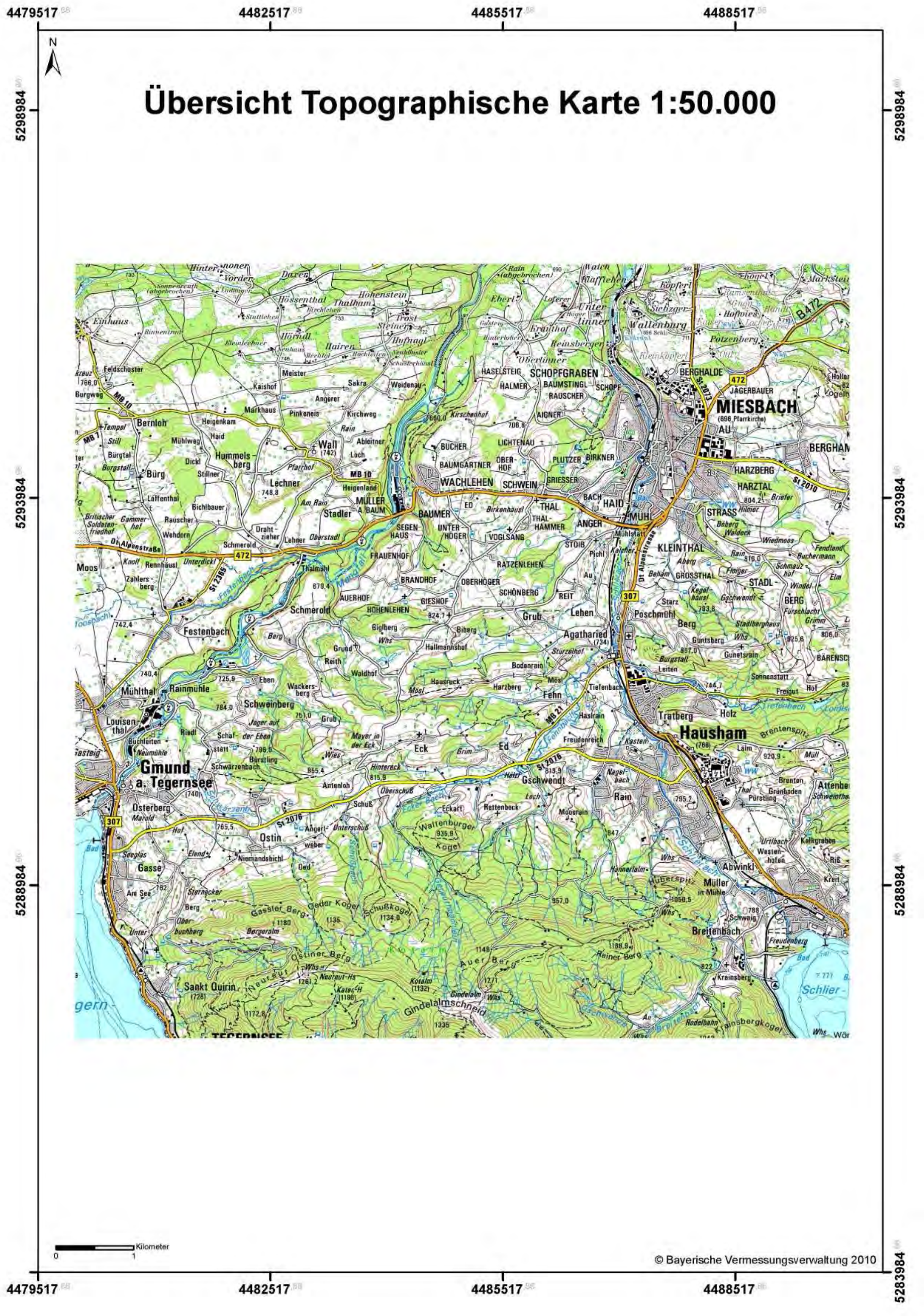
Anlage 2: Von OGR unterstützte Vektordatenformate (ogrinfo –formats)

ESRI Shapefile (read/write)
MapInfo File (read/write)
UK .NTF (readonly)
SDTS (readonly)
TIGER (read/write)
S57 (read/write)
DGN (read/write)
VRT (readonly)
REC (readonly)
Memory (read/write)
BNA (read/write)
CSV (read/write)
NAS (readonly)
GML (read/write)
GPX (read/write)
KML (read/write)
GeoJSON (read/write)
Interlis 1 (read/write)
Interlis 2 (read/write)
GMT (read/write)
SQLite (read/write)
ODBC (read/write)
PGeo (readonly)
OGDI (readonly)
PostgreSQL (read/write)
MySQL (read/write)
XPlane (readonly)
AVCBin (readonly)
AVCE00 (readonly)
DXF (read/write)
Geoconcept (read/write)
GeoRSS (read/write)
GPSTrackMaker (read/write)
VFK (readonly)

Anlage 3: Übersichtskarte Digitale Orthophotos 20cm Bodenauflösung



Anlage 4: Übersichtskarte Topographische Karte 1:50.000



Anlage 5: Getestete MapServer UMN Ausgabeformate

```
OUTPUTFORMAT
  NAME "jpeg"
  DRIVER "AGG/JPEG"
  MIMETYPE "image/jpeg"
  IMAGEMODE RGB
  FORMATOPTION QUALITY=90
END

OUTPUTFORMAT
  NAME "jpeg"
  DRIVER "AGG/JPEG"
  MIMETYPE "image/jpeg"
  IMAGEMODE RGB
  FORMATOPTION QUALITY=80
END

OUTPUTFORMAT
  NAME "png"
  DRIVER "AGG/PNG"
  MIMETYPE "image/png"
  IMAGEMODE RGB
END

OUTPUTFORMAT
  NAME "png"
  DRIVER "AGG/PNG"
  MIMETYPE "image/png"
  IMAGEMODE RGB
   FORMATOPTION "QUANTIZE_FORCE=ON"
END

OUTPUTFORMAT
  NAME "gif"
  DRIVER "GD/GIF"
  MIMETYPE "image/gif"
  IMAGEMODE PC256
END
```

Anlage 6: Verwendete Verkehrszeichen

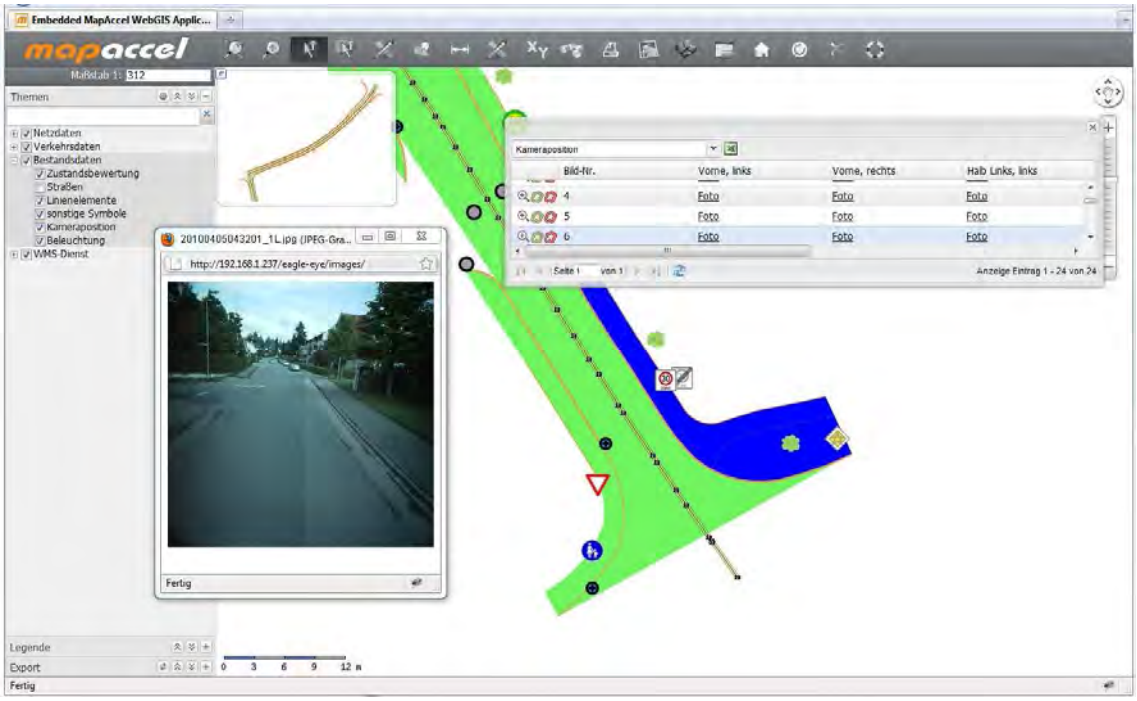
Quelle: THUBAUVILLE (1999)

Bezeichnung	**Nummer**	**Bild**
Gefahrstelle	101	
Kreuzung oder Einmündung mit Vorfahrt von rechts	102	
Kurve (links)	103-10	
Kurve (rechts)	103-20	
Doppelkurve (zunächst links)	105-10	
Doppelkurve (zunächst rechts)	105-20	
verengte Fahrbahn	120	
Baustelle	123	
Stau	124	
Gegenverkehr	125	
Vorfahrt gewähren!	205	
Halt! Vorfahrt gewähren!	206	
vorgeschriebene Vorbeifahrt – links vorbei	222-10	
vorgeschriebene Vorbeifahrt – rechts vorbei	222-20	
Haltestellen Straßenbahnen oder Linienbusse	224-50	

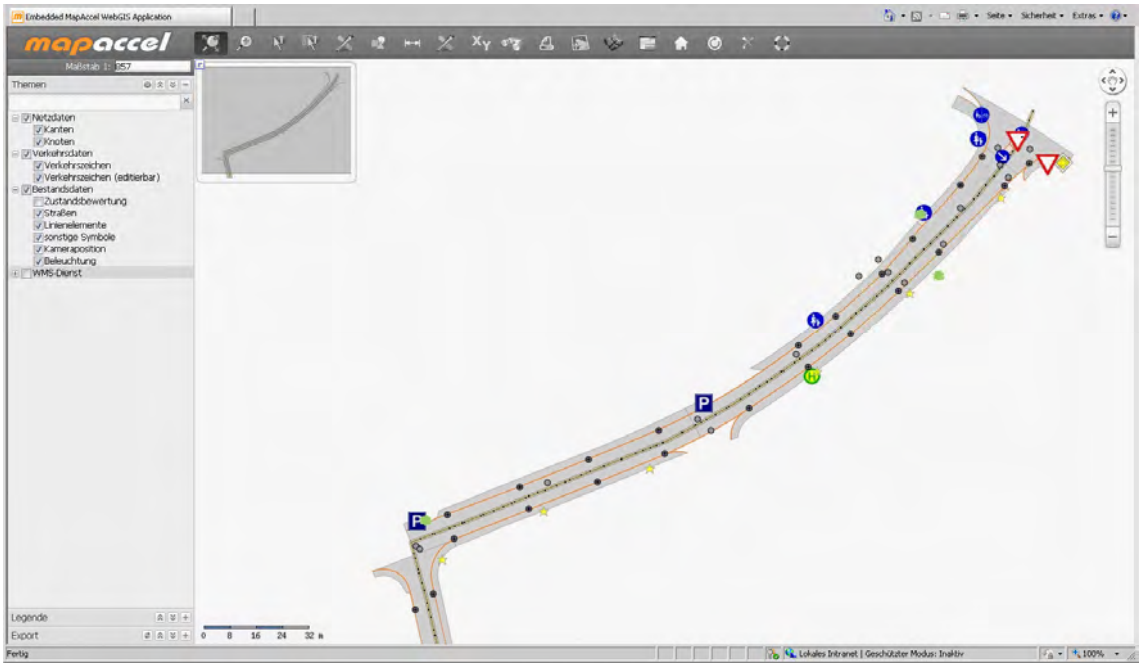
Fußgänger	239	
gemeinsamer Fuß- und Radweg	240	
getrennter Rad- und Fußweg	241-30	
getrennter Fuß- und Radweg	241-31	
Verbot für Fahrzeuge aller Art	250	
Verbot der Einfahrt	267	
Beginn der Zone mit zulässiger Höchstgeschwindigkeit	274.1-50	
Ende der Zone mit zulässiger Höchstgeschwindigkeit (einseitig)	274.2-50	
Beginn/Ende der Zone mit zulässiger Höchstgeschwindigkeit (doppelseitig)	274.2-40	
Haltverbot (Anfang)	283-10	
Haltverbot (Ende)	283-20	
Haltverbot (Mitte)	283-30	
Haltverbot (ohne Richtungspfeile)	283-50	
eingeschränktes Haltverbot (Anfang)	286-10	
eingeschränktes Haltverbot (Ende)	286-20	
eingeschränktes Haltverbot (Mitte)	286-30	

eingeschränktes Haltverbot (ohne Richtungspfeile)	286-50	
Vorfahrtstraße	306	
Ende der Vorfahrtstraße	307	
Parkplatz	314	

Anlage 8: Screenshots des WebGIS Portals mit den Straßenbefahrungsdaten



Aufruf von Befahrungsbildern



Darstellung der Flächengeometrien