Praxishandbuch

WebGIS mit Freier Software

UMN MapServer

PostgreSQL/PostGIS

AveiN!

Mapbender

Ein Gemeinschaftsprojekt von CCGIS und terrestris

CCGIS GbR
www.ccgis.de

terrestris GbR
www.terrestris.de

Änderungsvorschläge, Kritik und Reaktionen sind willkommen. Kontakt: a.trakas@ccgis.de

Inhaltsverzeichnis

Vorwort

Dieses Handbuch wurde in Zusammenarbeit der Firmen CCGIS GbR und terrestris GbR erstellt.

WebGIS-Anwendungen, Freie Software, Open Source, WMS, OGC sind nur einige wenige Begriffe aus einer langen Liste an Schlagwörtern, die immer häufiger im Zusammenhang mit Geographischen Informationssystemen (GIS) genannt werden. Es sind wichtige Begriffe, hinter denen sich verschiedene Möglichkeiten der Architekturen und Funktionalitäten von GI-Systemen verbergen.

Mit dem vorliegenden Handbuch wollen wir zur Lichtung des Begriffsdschungels und zum Verständnis der Möglichkeiten beitragen, die sich auf Grund der Neuerungen auf dem GIS-Markt ergeben haben.

Dieses Handbuch versteht sich nicht als Nachschlagewerk oder als vollständige Bedienungsanleitung einzelner WebGIS Komponenten, sondern dient als Leitfaden und Sammlung von Informationen zu Architekturen mit Freie Software Komponenten im geplanten Einsatz in heterogenen GI- und Geodateninfrastrukturen.

Die Funktionsweisen der hier vorgestellten Software werden nicht en detail erklärt, hierzu gibt es eine umfangreiche Liste diverser Links.

Die nachfolgenden Kapitel bilden den momentanen Stand der Entwicklung ab und bieten daher auch lediglich einen aktuellen Überblick. Da die beschriebenen Bereiche einer stetigen Dynamik unterliegen, wird sich dieses Handbuch auch weiterentwickeln. Wir fordern Sie auf, sich daran zu beteiligen: Anregungen, Kritik oder Beiträge zum Buch sind herzlich willkommen.

Nun wünschen wir viel Spaß beim Lesen und stehen Ihnen natürlich für Fragen zum Thema gerne zur Verfügung.

Bonn, im März 2004

Einleitung

von Arnulf Christl & Athina Trakas

In vielen öffentlichen Verwaltungen und Unternehmen in Deutschland wird bereits heute Open Source und Freie Software eingesetzt, nur ist dies nicht immer bekannt. Wie einige prominente Beispiele zeigen (z.B. Schwäbisch Hall, München) wird der Einsatz von Open Source und Freier Software immer häufiger auch von der Tagespresse wahrgenommen. Allerdings meist nur, wenn er medienwirksam als "Kampf" mit Microsoft in Szene gesetzt wird. Dabei werden die Argumente oft auf geringere Kosten beim Einsatz von Open Source reduziert, was 1. nicht unbedingt stimmen muss und 2. lediglich ein Argument unter vielen ist.

In diesem Kapitel werden daher zunächst die grundsätzlichen Konzepte *Freie Software* und *Open Source* vorgestellt. Darauf aufbauend werden im Kapitel *Der Einsatz von Open Source und Freier Software in Geodateninfrastrukturen* Probleme und Hindernisse bei der Anwendung dieser Software-Konzepte in der öffentlichen Verwaltung identifiziert und analysiert. Standards und Normen rund um das Open GIS Consortium werden im Kapitel *OGC und WMS Basis Know-How* erläutert.

Die weiteren Kapitel (*Einführung in WebGIS und in die Funktionsweise des UMN MapServer, Datenhaltung mit PostgreSQL/PostGIS, Schnelles Erstellen einfacher Karten (AveiN! und andere Oberflächen), Kompilieren und Betrieb auf verschiedenen Plattformen, UMN MapServer als OGC-konformer Web Map Service, Mapbender und UMN MapServer*) beschreiben die Architekturkomponenten UMN MapServer, PostgreSQL/PostGIS, AveiN! und Mapbender näher.

In einem weiteren Kapitel werden einige *Beispiele* für den Einsatz freier WebGIS Architekturen aus der Praxis vorgestellt. Abschließend werden die Erkenntnisse nochmals zusammengefasst und weitere Vorteile für Open Source und Freie Software jenseits der Kostendiskussion angeführt.

1. Begrifflichkeiten

Das Thema Freie und Open Source Software[1] hat eine lange Tradition und teilt sich in verschiedene Kategorien und Bereiche. Wenn Sie der Meinung sind, Freeware und Free Software bezeichnen die selbe Art von Software, wird ihnen die folgende Unterscheidung weiterhelfen. Um allerdings alle Auswirkungen und Implikationen von Freier Software und Open Source zu verstehen, ist es hilfreich oder sogar erforderlich, mehr als diese kurze Einführung zu lesen.

[1] Für die Unterscheidung der Worter Freie Software (FS) und Open Source Software (OSS) schauen sie bitte auch auf die Internetseiten der Free Software Foundation (http://www.fsf.org/philosophy/free-software-for-freedom.html) und der Open Source Initiative http://www.opensource.org/docs/definition.php. In diesem Artikel werden beide Wörter synonym verwendet oder in der zusammengesetzten Abkürzung OS/FS.

1.1. Open Source

Den vom Menschen lesbaren Teil einer Software nennt man den Source Code. In der traditionellen Softwareentwicklung muss der Source Code nomalerweise in eine Binärdatei kompiliert werden. Diese ist dann von Maschinen lesbar und kann vom Computer ausgeführt werden. Der Begiff Open Source bezeichnet Software, bei der der Source oder Quell Code lesbar ist, mitgeliefert wird und frei weitergegeben werden kann.

Hersteller und Entwickler proprietärer Software behandeln in den meisten Fällen den Source Code wie ein Geheimnis und das einzige Gut, um sich gegen potentielle Konkurrenten durchsetzen zu können.

Trotzdem ist es mit relativ einfachen Mitteln, möglich Software zu dekompilieren und diese bis ins Detail zu studieren. Da man meist eine Kopie der Software benötigt, um damit arbeiten zu können, ist es sehr schwer oder beinahe unmöglich, Software von der technischen Seite als Geheimnis zu bewahren. Daher sind Softwarelizenzen proprietärer Programme sehr strikt, was den Umgang und Gebrauch mit den Software betrifft. Zusätzlich enthalten sie einige Barrieren, die den Gebrauch der Software einschränken können. Das heisst, es wird mit viel Aufwand versucht, eine vom Prinzip her offene Technologie zu verschlüsseln und den Nutzen zu beschränken.

Der Begriff Open Source beschreibt hingegen explizit, dass die Sourcen (Quellen) einer Software zugänglich sein müssen. Die Open Source Initiative (OSI) erweitert die Offenheit auf andere Bereiche, die sich meist auch in der Free Software Bewegung finden. Open Source ist eine Voraussetzung für Freie Software.

1.2. Freie Software

Das Konzept der Freien Software, von Richard Stallman im Jahr 1984 initiiert, betrifft neben der Offenheit des Quellcodes auch philosophische und theoretische Fragen des Copyright, Copyleft und der Nutzerrechte. Entstanden ist das Konzept aus dem einfachen Bedarf eines freien, UNIX-ähnlichen Betriebssystems, das GNU is Not UNIX (GNU) genannt wurde.

Einige mögen denken, dass “Frei” im Kontext Freier Software wie “Frei” in “Freibier” verstanden werden kann. Allerdings sollte “Frei” stattdessen verstanden werden wie das “frei” in freier Rede oder Freiheit. Freie Software bedeutet die Freiheit des Nutzers, die Software zu nutzen, sie zu kopieren, weiterzuverteilen, zu studieren, sie zu ändern und zu verbessern. Genauer, Freie Software beinhaltet vier Freiheiten des Nutzers:

- Die Freiheit, das Programm für jeden Zweck zu benutzen (Freiheit 0).
- Die Freiheit, zu verstehen, wie das Programm funktioniert und wie man es für seine Ansprüche anpassen kann (Freiheit 1). Der Zugang zum Quellcode ist dafür Voraussetzung.

- Die Freiheit, Kopien weiterzuverbreiten, so dass man seinem Nächsten weiterhelfen kann (Freiheit 2).
- Die Freiheit, das Programm zu verbessern und die Verbesserungen der Öffentlichkeit zur Verfügung zu stellen, damit die ganze Gemeinschaft davon profitieren kann (Freiheit 3). Der Zugang zum Quellcode ist dafür Voraussetzung (http://www.fsf.org/philosophy/free-sw.html).

Damit Software in diesem Sinne frei ist, muss ihr Source Code ohne Restriktionen verfügbar sein.

1.3. Open Source / Freie Software

Für eine einfachere Handhabung wird in diesem Handbuch der Begriff Open Source / Freie Software oder abgekürzt OS/FS verwendet, wenn Freie Software und Open Source Software beschrieben wird. Wie bereits erwähnt: für eine genaue Unterscheidung der beiden Begriffe sind die jeweiligen Internetseiten der Free Software Foundation oder der Open Source Initiative zu besuchen.

Für die meisten Nutzer Freier Software sind momentan die ökonomischen Aspekte die reizvollsten Vorteile. Außer den modernen und akzeptierten Modellen zur Softwareentwicklung, kann OS/FS in soziale und marktrelevante Dimensionen ausgedehnt werden.

Nach Meinung des oekonux-Projekts[2] erklärt sich das hohe Potential von OS/FS dadurch, dass Eingriffe in die Entwicklung und Beschränkungen der Software aufgrund von Überlegungen einer Marketingabteilung entfallen. Der Schwerpunkt des Interesses liegt dagegen ausschließlich auf der Qualität der Software. Auf lange Sicht wird lediglich die Qualität der Software darüber entscheiden, ob sie durch die Nutzergemeinschaft angenommen wird oder nicht (Frankfurter Rundschau vom 06.12.2003 / Spaß am Programmieren).

Die Einstellung der Nutzer zu Freier Software ändern sich in der Regel, je länger sie sich mit OS/FS beschäftigen und diese nutzen. Dies betrifft nicht nur die Sicht der Nutzer, sondern gilt auch für Unternehmen und Softwarehersteller. Anfangs und vor allem heute stehen oft monetäre Argumente für den Einsatz von OS/FS im Vordergrund. Wenn Nutzer beginnen, sich mit den Softwarekonzepten auseinanderzusetzen, beginnen sie sehr schnell weitere Aspekte zu schätzen, wie z.B. die hohe Qualität, die Sicherheitsaspekte oder die konstante Weiterentwicklung der Software.

2. Übernahme von OS/FS Konzepten duch die Nutzer

Die Einführung der Konzepte von Freier und Open Source Software wurde von einigen Softwarespezialisten initiiert, die den Zugang zum Quellcode benötigten, um Software an

[2] In diesem Projekt werden ökonomische und politische Eigenschaften Freier Software, unter der Verwendung verschiedener Methoden, betrachtet sowie diskutiert (http://www.oekonux.org/ vom 17.12.2003).

ihre speziellen Bedürfnisse anpassen zu können. Eine bunte Community, die über das Internet kommunizierte, begann sich zu entwickeln. Diese Gruppe war nicht auf zentrale Institutionen angewiesen, sondern setzte das Konzept “Wissen multiplizieren durch die Verbreitung von Wissen” um.

Erstaunlicherweise werden OF/FS Konzepte auch immer mehr zu einem wichtigen Aspekt für Regierungen, obwohl diese Konzepte ein sehr dezentralisiertes und dereguliertes Konzept widerspiegeln. So wird z.B. in Deutschland Linux und andere OS/FS auf Bundes-, Landes- und kommunaler Ebene eingesetzt. Staatssekretär Siegmar Mosdorf argumentiert, dass Freie Software die Sicherheit von Software verbessert[3]. Der deutsche Innenminister Otto Schilly erklärt, dass die Regierung durch den Einsatz von FS/OS seine Abhängigkeit von einem einzelnen Anbieter verringert, mit gleichzeitiger größerer Diversität im Bereich des Computereinsatzes.

Im Dezember 2003 stellte die Europäische Kommission, Generaldirektion Informationsgesellschaft einen neuen Bereich auf ihren Internetseiten zum Thema Freie Software und Open Source in Programmen und Initiativen der EU bereit. Das Open Source Observatory (OSO) initiiert von der Generaldirektion Unternehmen und gegründet unter dem IDA Programm (Interchange of Data between Administrations), hat sich zum Ziel gesetzt, einen verständlichen Überblick über Aktivitäten im Bereich Open Source in aktuellen und künftigen Mitgliedstaaten der EU zu bieten[4].

2.1. Nicht überall wo „open“ draufsteht ist auch OS/FS drin

Wie viele erfolgreiche Begriffe, findet sich auch der Zusatz „open“ in vielen Kombinationen, je nach Laune der Marketingabteilungen. Daher gibt es sehr viele Begriffe, die mit einer Offenheit werben, die nicht den OS/FS Konzepten entsprechen. Es ist daher erforderlich, genau hinzuschauen, was nun mit „open“ oder „frei“ gemeint ist.

Die in den folgenden Kapiteln vorgestellten Architekturkomponenten für WebGIS-Systeme sind alle Freie oder Open Source Software.

2.2. Proprietäre versus kommerzielle Software

Die beiden Begriffe proprietär und kommerziell werden ebenfalls gerne in der „Nachbarschaft“ von Freier und Open Source Software verwendet – als Gegenpart hierzu. Dies ist allerdings begrifflich nicht ganz korrekt, wie die folgenden Erläuterungen zeigen.

2.2.1.Proprietärere Software

Unter proprietärer Software wird nach der Free Software Foundation

[3] Heise (29. Juni 2000): Wirtschaftsminister: Sicherheit und Innovation durch Open Source. Internetrecherche vom 16. Dezember 2003: http://www.heise.de/newsticker/data/odi-29.06.00-000/

[4] Die Seiten können unter folgenden URLs gefunden werden:
Informations Society: http://europa.eu.int/information_society/activities/opensource/text_en.htm,
OSO: http://europa.eu.int/ISPO/ida/jsps/index.jsp?fuseAction=home

(http://www.gnu.org/philosophy/categories.de.html) Software verstanden, die weder frei noch halbfrei ist. Veränderungen oder die Weiterverbreitung sind verboten oder verlangen, dass eine Erlaubnis dafür vorliegt. Diese Erlaubnis ist oft so stark eingeschränkt, dass die Software effektiv nicht verändert oder verbreitet werden darf.

2.2.2. Kommerzielle Software

Software, die mit dem Ziel entwickelt wurde, aus der Nutzung dieser Software Geld zu machen wid gemeinhin als kommerzielle Software bezeichnet. "Kommerziell" und "proprietär" ist allerdings nicht synonym zu verwenden! Viele kommerzielle Softwareprodukte sind zwar proprietär, aber es gibt auch kommerzielle Freie Software, und es gibt nichtkommerzielle unfreie Software.

Beispiel:

GNU Ada, zum Beispiel, wird immer unter den Bedingungen der GNU GPL vertrieben und jede Kopie ist freie Software; aber ihre Entwickler verkaufen Supportverträge. Wenn ihre Vertreter mit voraussichtlichen Kunden reden, sagen die Kunden manchmal "Wir würden uns mit einem kommerziellen Compiler sicherer fühlen." Die Vertreter antworten: "GNU Ada *ist* ein kommerzieller Compiler; zufälligerweise ist er auch freie Software." (http://www.gnu.org/philosophy/categories.de.html).

Folgendes Schaubild verdeutlich nochmals die Einordnung von kommerziell und proprietär in die unterschiedlichen Softwarearten. Aufgrund der oben beschriebenen Unterschiede ist es wichtig, die beiden Begriffe nicht synonym zu verwenden.

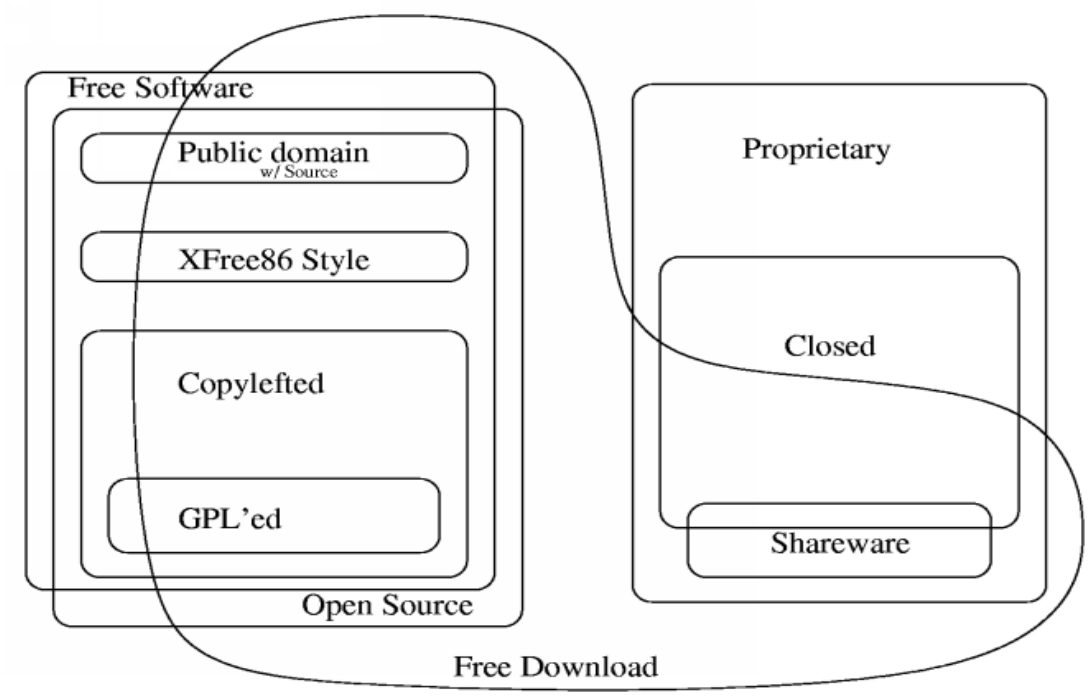

Abb.: Proprietäre, Freie und Open Source Software aus: http://www.gnu.org/philosophy/categories.de.html

* * *

1. Der Einsatz von OS/FS in Geodateninfrastrukturen

von Arnulf Christl

GDI (Geodateninfrastrukturen) sind in aller Munde. Der Begriff gewinnt mit zunehmender Datendichte, einer immer besser vernetzten Wissensgesellschaft und immer höheren Funtkionstiefe der Softwarepakete zunehmend an Bedeutung. GIS wird schon lange nicht mehr als isolierte Softwarelösung betrieben, sondern hat sich in eine Bezeichnung für eine Richtung der Informationstechnologie aufgelöst. Das GIS als Selbstzweck ist tot – es lebe die GDI (und ein Teil davon ist auch "klassisches" GIS).

Internet & Webtechnologie können wesentliche Bestandteile beim Aufbau einer GDI sein. Sie sind neben Betriebssystemen die Bereiche, in denen sich OS/FS Lösungen auch besonders gut entwickeln konnten und können. Die Problembereiche beim Aufbau einer wie auch immer gearteten GDI sind sehr vielfältig, neben den offensichtlichen technischen Fragen gibt es eine Reihe historisch bedingter Rahmenbedingungen und nicht zuletzt hängt es an Personen, Arbeitskreisen, aber auch Verwaltungsvorschriften und Empfehlungen, die den Rahmen weiter abstecken.

1.1 Problem oder Ziel?

Der Aufbau einer GDI kann zentral gesteuert und koordiniert erfolgen, oder sich dynamisch entwickeln. Eine Mischung aus beiden Ansätzen erscheint am erfolgversprechendsten.

1.1.1. Statischer Ansatz

Alle Fragestellungen werden als Probleme definiert, für die eine (meist technische) Lösung gefunden werden muss. Um eine Kompatibilität der einzelnen an der GDI beteiligten Komponenten zu gewährleisten (was der zentrale Sinn einer Geodateninfrastruktur ist), müssen alle Softwarelösungen zum Zeitpunkt der Planung bereits bekannt sein und im besten Fall zumindest als Demo verfügbar sein. Problematisch bei diesem Ansatz ist, dass nur vom aktuellen Stand der Technik und des Wissens ausgegangen werden kann und neue Lösungen, die evtl. erst während der Realisierungsphase der GDI Produktreife erlangen, nicht berücksichtigt werden können. Da sich der Aufbau einer GDI je nach Anzahl der Beteiligten, Menge der Daten und Diversität der Zielgruppen über mehrere Jahre erstrecken kann, werden durch den statischen Ansatz Innovationen verhindert.

1.1.2. Dynamischer Ansatz

Wenn die einzelnen Komponenten der Infrastruktur unter Berücksichtigung der einschlägigen Normen und Standards entwickelt wurden, lassen sie sich ohne großen Aufwand zusammenschließen, obwohl sie möglicherweise völlig unabhängig voneinander entwickelt wurden. Dies eröffnet ein wesentlich größeres Spektrum an möglichen

Lösungen, die als einzige Bedingung die vorgegebenen Normen und Standards einhalten müssen. Hierzu zählen im GIS Kontext vorrangig die Spezifikationen des OGC (z.B. WMS und WFS, s. Kapitel OGC & WMS) und ISO Normen für Geodaten, Metadaten und Dienste (z.B ISO 19115 und ISO 19119, www.iso.ch).

2. Die GDI Architektur als Leitbild und offene Zieldefinition

Alternativ zum Problemansatz kann der Aufbau einer GDI auch als Rahmenvorgabe in Form eines Leitbildes verstanden werden, welches sich dynamisch entwickelt und mit den Anforderungen wächst. Auch Anforderungen an die Projektsteuerung werden durch solch einen Ansatz neu definiert und benötigen Freiheiten, die z.B. in einer öffentlichen Verwaltung wesentlich schwieriger durchzusetzen sind als in der privaten Wirtschaft. Deshalb haben sich diese Konzepte auch zuerst bei privatwirtschaftlich agierenden Unternehmen durchgesetzt.

Viele großangelegte Softwareprojekte sind schlicht aufgrund ihrer Größe gescheitert, neuestes prominentes Beispiel ist die deutsche Mautlösung, die als monolithisches Konzept aufgesetzt wurde und jetzt vollkommen handlungsunfähig ist, weil einige Komponenten nicht zusammenspielen. Das Problem ist nicht die "Unfähigkeit" der Programmierer, sondern ein für das Problem nicht geeigneter Softwareentwicklungsansatz. Das erklärt auch Aussagen wie: "Sie müssen bedenken, dass es sich dabei um eine sehr, sehr komplexe Software handelt"; als ob das in irgendeiner Form ein Erfolgs- oder Leistungskriterium wäre. Das ist es nicht, sondern es ist schlicht der falsche Ansatz[5].

2.1. Lernen aus der Softwareentwicklung

Freie Software Entwicklungskonzepte spiegeln eine zieloffene Vorgehensweise wider. Das Ergebnis einer zieloffenen und dynamischen Vorgehensweise wird immer auch den aktuellen Stand der Technik widerspiegeln. Alle Fragestellungen werden auf kleinere Unterprojekte verteilt und dort mit einem kleinen Team von Beteiligten gelöst. Das ist genau das "Geheimnis" des Erfolgs von OS/FS. Nicht zuletzt deshalb werden diese Konzepte zunehmend auch von proprietären Herstellern gewählt und bei der Softwareentwicklung berücksichtigt.

Im folgenden wird der Entwicklungsablauf eines solchen Teilprojektes unter Nutzung der OS/FS Technologien beschrieben.

2.1.1. Alternative A

- Ein lokales, klar definiertes, überschaubares Problem wird identifiziert.
- Für die Lösung dieses Problems wird eine technische Lösung erarbeitet.

[5] Dr. Egbert Meyer; http://www.heise.de/mobil/artikel/2002/11/18/maut/default.shtml

- Es gibt bereits eine technische Lösung für das Problem (es gibt keine: siehe unten).
- Sie wird übernommen. Das Problem ist gelöst und der Prozess stirbt.

2.1.2. Alternative B

- Es gibt noch keine technische Lösung für das Problem.
- Eine technische Lösung für die Lösung des Problems kann erarbeitet werden (die Finanz- und Personalressourcen stehen zur Verfügung).
- Das Problem ist gelöst und der Prozess stirbt.

Anmerkung: Bei OS/FS Konzepten würde diese Problemlösung auch als Open Source Projekt der Allgemeinheit zur Verfügung gestellt werden. Die technische Unterstützung erfolgt z.B. über SourceForge, eine der Plattformen, die eine Infrastruktur für den Betrieb von Software Projekten bereitstellten. Dies bindet andere Nutzer in die Pflege und Weiterentwicklung der Software ein. Dadurch verbessert sich bei geteilten Entwicklungskosten die Qualität der Software.

2.1.3. Alternative C

- Die technische Lösung ist nicht mit eigenen Ressourcen, egal welcher Form, zu realisieren.
- Es werden andere Interessierte gesucht, die das gleiche Problem haben.
- Diese organisieren sich (temporär) zu einer Interesengemeinschaft, um genügend Ressourcen (Personen, Infrastruktur, Finanzen) für die Lösung des Problems bereitzustellen.
- Die Lösung wird allgemein zugänglich gemacht und steht anderen Anwendern zur Verfügung.

Bei Freie Software Projekten kann eine solche Interesengemeinschaft die ganze Welt umspannen. Eines der bekanntesten Beispiele neben Linux ist z.B. das "http Server-Project“ der Apache Software Foundation – 60% aller Webserver weltweit nutzen den Apache Web Server. Gerade aufgrund der hohen Relevanz von Sicherheitsfragen beim Betrieb von Webservern ist dieser Bereich für alle Webadministratoren von sehr hoher Bedeutung und wird entsprechend intensiv getestet.

2.2. Einhaltung von Standards

Ein ganz wichtiges Kriterium bei dieser Vorgehensweise ist, dass alle Komponenten immer mit der Maßgabe bester Kompatibilität entwickelt werden. Relevante Standards müssen unbedingt eingehalten werden. Hierzu zählen im GIS Kontext (wie oben bereits ausgeführt) vorrangig die Spezifikationen des OGC (z.B. WMS und WFS) und ISO Normen für Geodaten, Metadaten und Dienste (z.B. ISO 19115 und ISO 19119).

3. Rollen und Aufgaben der Projektbeteiligten

Der Ansatz einer dynamischen Zielorientierung bei der Planung und dem Aufbau einer GDI reflektiert sich im Selbstverständnis der Beteiligten und hat Auswirkungen auf die Vorgehensweise bei Ausschreibungen und auf die Beauftragung von Teilprojekten. Dieser Ansatz ermöglicht es auch, dass unterschiedliche Unternehmen zur Problemlösung beitragen können, was Kosten reduzieren und den Produktionsbeginn beschleunigen kann. Folgende Personengruppen sind beim Aufbau einer Geodateninfastruktur beteiligt:

- Entscheider
- Koordinatoren
- Techniker
- Anwender

Je nachdem, ob ein Top-Down Ansatz oder eine problemorientierte Vorgehensweise verfolgt wird, sind die Aufgaben der Beteiligten sehr unterschiedlich bewertet.

3.1. Top-Down Methode

Bei der Top-Down Vorgehensweise wird auf Entscheiderebene eine Architektur vorgegeben, die durch die Koordinatoren verfeinert, von den Technikern umgesetzt und nach Abschluss des Projektes den Anwendern vorgesetzt wird. Von Entscheiderebene aus betrachtet, scheint das zunächst die einzig sinnvolle Vorgehensweise zu sein. Die Erfahrung zeigt jedoch, dass solche Projekte an den Bedürfnissen der Anwender vorbeizielen können und schlicht keine Verwendung finden. Erschwerend kommt hinzu, dass die Dauer großer Projekte sich über einen wesentlich längeren Zeitraum erstrecken kann, als die Entwicklungszyklen der eingesetzten Software. Das Ergebnis sind veraltete technische Lösungen, die im besten Fall einem aktuellen Kosten/Nutzen Vergleich nicht standhalten, im schlimmsten Fall aber bereits bei der Einführung (also der Freischaltung für die Anwender) eine veraltete Architektur benötigen.

3.2. Problemorientiertes Vorgehen

Bei einem problemorientierten Ansatz können Architekturen zu einem Gesamtsystem zusammenwachsen. Aus jeder Problemstellung entstehen Lösungen, die sich zu einem großen Ganzen zusammenfügen. Wichtige Voraussetzung ist dabei die Einhaltung von Standards und die Offenheit des Systems.

3.3. Entscheider

Ein Entscheider sollte sich nicht vorab für **ein** System entscheiden müssen. Eine Entscheidung sollte immer nur für einen überschaubaren Teilbereich gefällt werden. Je

größer ein Gesamtsystem ist, um so schwieriger bis unmöglich wird ein vorgeschalteter Entscheidungsprozess. OS/FS Konzepte bieten in Verbindung mit einem klaren Bekenntnis zu offenen Schnittstellen optimale Rahmenbedingungen für eine problemorientierte Vorgehensweise.

Die Aufgabe des Entscheiders verschiebt sich weg von einer unumkehrbaren, absoluten Vorab-Entscheidung hin zu einer projektbegleitenden Beratung oder Überwachung der einzelnen Prozesse. Die Entscheidungen werden problemorientiert auf Architekturebene getroffen und legen Standards und Schnittstellen fest. Das beinhaltet nicht nur technische Schnittstellen, sondern auch die Koordination, wer wann mit wem zusammenarbeitet.

3.4. Koordinatoren

Der Projektleiter oder Koordinator konkretisiert die Leitbilder, die von den Entscheidern vorgegeben werden, auf einzelne Arbeitspakete, koordiniert die Abfolge und überwacht den Fortschritt der einzelnen Projekte, um gegebenenfalls zu intervenieren, wenn Schwierigkeiten auftreten.

Eine weitere Aufgabe der Projektleiter ist, die Kommunikation zwischen Entscheider und Techniker zu gewährleisten. Dabei ist es wichtig, dass diese Kommunikation bidirektional ist. Ein Techniker kann durch den Einblick in die Softwarearchitektur gegebenenfalls verhindern, dass unsinnige Lösungen an Stellen eingesetzt werden, die bereits mit einer funktionierenden Lösung ausgestattet sind, nur weil die Gesamtarchitektur an dieser Stelle den Einsatz einer anderen Technologie vorsieht.

3.5. Techniker

Die Techniker setzen nicht nur die Vorgaben der Koordinatoren technisch um, sondern beraten sowohl die Projektleiter als auch Entscheider auf technischer Ebene. Techniker sollten aus diversen, möglichst heterogenen Bereichen rekrutiert werden, um "inzestuöse Entwicklungen" zu vermeiden. Das ist kein lustiges Wortspiel, sondern eine wichtige Maßnahme, um zu verhindern, dass in speziellen Gebieten unsinnige Architekturen implementiert werden, obwohl lediglich (fremdes oder neues) Know-How zur Lösung fehlt. Dabei ist es sicherlich von Vorteil, wenn auch konträre Meinungen diskutiert werden können.

3.6. Anwender

Die Anwender stellen die mit Abstand wichtigste Gruppe beim Aufbau einer Infrastruktur dar. Das wird oft verkannt, mit dem Ergebnis, dass Systeme aufgesetzt werden, die nicht so funktionieren wie es für die Anwender erforderlich wäre und in der Folge einfach nicht genutzt werden. Auch hier bietet der OS/FS Ansatz ideale Mechanismen, um diesem Problem entgegenzuwirken. Die direkte Integration der Endanwender in den Entwicklungsprozess ermöglicht die mit Abstand beste Qualitätssicherung. Kein noch so

ausgefeiltes Testsystem ist in der Lage, die Anwender und deren Fragestellungen zu 100% abzubilden. Da die Anwender ja direkt einbezogen werden können und gar nicht simuliert werden müssen, verursacht dieses Vorgehen praktisch keinen zusätzlichen Aufwand.

Die frühzeitige Einbindung der Anwender in den Entwicklungsprozess bedeutet auch, dass sie sich schrittweise mit dem System vertraut machen können. Das reduziert den Schulungsaufwand und minimiert Reibungsverluste bei der Umstellung oder Einführung der neuen Komponenten.

3.7. Externe Projektbeteiligte

Neben den internen Projektbeteiligten gibt es oft Externe, die in den Prozess eingebunden sind. Diese können ebenfalls in allen oben genannten Gruppen vertreten sein. So kann ein beratender Techniker bereits bei den ersten Architekturentscheidungen wesentliches Know-How beisteuern.

3.8. Entwickler / Programmierer

Die Gruppe der Entwickler und Programmierer hat bei OS/FS einen sehr wichtigen Stellenwert. Entwickler können sowohl innerhalb des Projektes als auch außerhalb angesiedelt sein. Der Unterschied liegt dabei hauptsächlich in der Weisungsbefugnis, die bei den relativ flachen Strukturen einer projektorientierten Entwicklung nicht so stark ins Gewicht fällt.

Die Gruppe der Entwickler darf nicht zu stark von den Anwendern isoliert werden. Eine direkte Kommunikation zwischen Anwendern und Entwicklern kann dann problematisch werden, wenn die Entwickler mit Fehlermeldungen und Anfragen überschüttet werden. Es gibt eine Reihe von Techniken aus der freien Softwareentwicklung, die diese Kommunikation strukturieren können. Da diese Methoden auch bei weltweiten Projekten sehr erfolgreich Anwendung finden, sollte es eine der vordringlichen Aufgaben der Projektleitung und Koordination sein, diese Steuerungselemente effektiv einzusetzen.

3.9. Methoden zur Koordinierung der Softwareentwicklung

Jedes Softwareprojekt mit einem OS/FS Ansatz sollte über technische Ressourcen verfügen, um die Zusammenarbeit zwischen Planung, Leitung, Entwicklung und Anwendung zu koordinieren. Zu diesen gehören Userlisten, Benutzerforen, Buglists und Change Request Tracker; als Informations- und Dokumentationstool haben sich WIKI Pages bewährt.

4. Strukturen der öffentlichen Verwaltungen bieten Vorteile beim Einsatz von OS/FS

Normalerweise haben öffentliche Verwaltungen ein schlechtes Image, wenn es darum geht, Softwareprojekte zu erarbeiten und zu betreiben. Dieses schlechte Image begründet sich darin, dass viele Projekte scheitern, nie den Prototypstatus verlassen oder jahrelang als Test- und Demoversionen betrieben werden. Die Probleme werden oft auf verkrustete Strukturen zurückgeführt und die Schuld auf schlechte Planung oder zu geringe Geldmittel geschoben.

Gerade der Bereich GIS bietet allerdings eine ausgezeichnete Basis, um Vorteile der öffentlichen Verwaltung effektiv zu nutzen. Die Vorteile gerade beim Einsatz von OS/FS im Bereich GIS liegen auf der Hand:

- Alle Verwaltungen haben ähnliche Anforderungen.
- Die Verwendung bestimmter Softwarekomponenten bringt einer öffentlichen Verwaltung keinen kommerziellen Vorteil gegenüber einer anderen Verwaltung. Gemeinsamen Aktivitäten steht also kein Konkurrenzgedanke im Weg.
- Alle öffentlichen Verwaltungen müssen sich an die gleichen zentralen Vorgaben halten.
- Viele Fragestellungen müssen übergreifend zwischen mehreren Kommunen gelöst werden.
- Eine Inwertsetzung von Geodaten ist Ziel jeder öV.
- Das Geodatenmanagement unterscheidet sich nur im Detail, da Basisdaten überall weitgehend gleich sind. Regionale Unterschiede sind weniger relevant als Unterschiede im Know-How und beim Kenntnisstand der beteiligten Personen.
- In den öffentlichen Verwaltungen steht ein enormes Entwicklerpotential zur Verfügung.
- Geteiltes Wissen ist einfach multiplizierbar, da die öV bereits sehr stark vernetzt sind.

4.1. Alle Verwaltungen haben ähnliche Anforderungen

Die Verwaltungshierarchie bildet eine Pyramide, in der die obersten Bereiche (Ministerien, Landesämter) Leitbilder aufstellen und lediglich grobe Vorgaben machen. Die Basis bildet eine enorme Anzahl von Anwendern, die tatsächlich täglich Software nutzen, um Fragestellungen zu bearbeiten. Da sich die Fragestellungen in den allermeisten Fällen stark ähneln, liegt es nahe, gemeinsame Lösungen zu finden oder zu schaffen.

Ein wichtiger Bestandteil in dieser Hierarchie bilden die Kataster- und Vermessungsämter, die in Deutschland (föderal) sehr heterogen aufgebaut sind, sich aber dennoch weitgehend auf Standards geeinigt haben. Diese Standards sind z.T. völlig veraltet und bedürfen einer dringenden Überarbeitung, diese kommt aber aufgrund der heterogenen Strukturen und Ausgangspositionen nur sehr schleppend voran. Bereits bestehende

Geodateninfrastrukturen (bestehend aus Daten, Anwender-Know-How, Softwarepaketen und möglicherweise langfristigen Verträgen) bilden zwar das Fundament aller zukünftiger GDI, behindern aber oft mangels Flexibilität eine Weiterentwicklung und Anpassung an neue Anforderungen. Fehlende Standardisierungsbestrebungen schlagen spätestens auch hier mit erheblichen zusätzlichen Kosten zu Buche.

Einige relevante Datenstandards, die bereits umfangreich genutzt werden:

- EDBS (Austauschformat für amtliche Kartenwerke)
- ALK (Automatisiertes Liegenschaftskataster) beinhaltet u.a. Flurstücke und Gebäude und bildet damit die grundlegenden geometrischen Basisdaten
- ALB (Automatisiertes Liegenschaftsbuch) beinhaltet u.a. Eigentümer und Nutzungsinformationen zu Flurstücken und Gebäuden
- Luftbilder (Orthophotos) werden flächendeckend von der Landesvermessung bereitgestellt, vielen Kommunen stehen inzwischen zusätzlich höher aufgelöste Luftbilder zur Verfügung
- ATKIS (Amtliches Topographisch-Kartographisches Informationssystem)
- Topographische Daten in verschiedenen Maßstäben
- zunehmend neue Informationsebenen, die z.T. bereits über Dienste bereitgestellt werden (Mobilfunksendestandorte, Schutzgebiete, Richtwertkarten, etc.)

Zusätzlich gibt es Fortführungsstandards für manche Daten (z.B. EDBS/BZSN für das ALK-Werk), die so etwas ähnliches wie einen Funktionsstandard festschreiben. Bei zukünftigen Lösungen werden Standards, die nicht nur die Formate beschreiben, sondern den Zugriff darauf definieren, eine immer stärkere Rolle spielen (z.B. ALKIS und NAS).

4.2. Die Verwendung bestimmter Softwarekomponenten bringt einer öffentlichen Verwaltung keinen kommerziellen Vorteil

Aus Sicht einer öffentlichen Verwaltung bietet die Festlegung auf eine bestimmte Software oder einen Softwarehersteller keinen direkt messbaren kommerziellen Erfolg. In manchen Fällen mag die lokale Wirtschaft davon profitieren, allerdings nur, wenn das Budget auch tatsächlich in die Region fließt und nicht in unbekannten Kanälen internationaler Konzerne versickert.

4.3. Alle öffentlichen Verwaltungen müssen sich an die gleichen zentralen Vorgaben halten

Es gibt keine einheitlichen Regelungen oder zentrale Vorgaben in den föderalen Strukturen der öV in Deutschland. Wer wann welche Softwarepakete einsetzt, ist nicht festgeschrieben. Allerdings gibt es sehr klare Vorschriften, z.B. über die Bereitstellung von Geoinformationen zur Einsicht für den Bürger (Informationspflicht) und auch sehr klare

Anforderungen an den flächendeckenden Aufbau einer digitalen Grundkarte.

Mit welcher Software, welchen Mitteln und unter Zuhilfenahme welcher zusätzlichen Dienstleistungen diese Vorgaben erfüllt werden, ist nicht vorgeschrieben – auch hier gilt: Das bessere System wird durchkommen.

4.4. Viele Fragestellungen müssen übergreifend zwischen mehreren öffentlichen Verwaltungen gelöst werden

In jeder Gemeinde, Stadt, jedem Kreis und Land gibt es Fragestellungen, die über das eigene Grenzgebiet hinaus gehen. Das gilt für sehr viele, sehr pragmatische Fragestellungen, angefangen von der Polizei, der Feuerwehr und den Rettungsdiensten, Katastrophenschutz, über Straßennetze, aber auch naturräumliche Objekte wie das Gewässernetz. Was passiert, wenn nicht alle Anrainer eines Gewässers flussauf und flussab zusammenarbeiten, kann jedes Jahr aufs Neue bei vielfältigen Überflutungen gesehen werden.

4.5. Eine Inwertsetzung von Geodaten ist Ziel jeder öffentlichen Verwaltung

Jede öffentliche Verwaltung, die Jahrzehnte in den Aufbau von Basisgeodaten investiert hat, wird diese möglichst effektiv in Wert setzen wollen. Dabei gilt, dass bei allen Grenzfragen immer auch die Nachbarlösung kompatibel sein muss, im schlimmsten Fall sogar auf höchster Ebene, also länderübergreifend und staatenübergreifend, was nicht nur Fragen der Sprache aufwirft, sondern auch der Datenqualität, der Bemessungsgrundlagen etc..

4.6. Räumliche Unterschiede sind weniger relevant als Unterschiede im Know-How

Ein Stadtstaat (z.B. Bremen) hat grundsätzlich andere Anforderungen als ein Flächenstaat (z.B. Mecklenburg-Vorpommern) und natürlich wird Schleswig-Holstein küstenrelevante Objekte verwalten, die in Hessen nicht zu finden sind. Die Basisdaten sind auch trotz dieser regionalen Unterschieden überall weitgehend gleich.

Anders verhält es sich mit dem Know-How, das einer öV mit dem eigenen Personal zur Verfügung steht. Gerade auch bedingt durch die föderalen Strukturen und dadurch fehlender zentraler Vorgaben, wurden sehr unterschiedliche Schwerpunkte gesetzt. Über die letzten Jahrzehnte sind einige Softwarepakete gestorben und ganz neue dazu gekommen, das Know-How musste sich zwangsläufig weiterentwickeln. Inzwischen ist das Potential an Know-How sehr differenziert und inhomogen verteilt.

4.7. In den öffentlichen Verwaltungen steht ein enormes Entwicklerpotential zur Verfügung

In jeder noch so kleinen öffentlichen Verwaltungseinheit gibt es jemanden, der sich mehr oder weniger gut mit Software auskennt. Dieses Know-How sollte gepflegt werden. Dort wo dies bereits geschieht, rechnet sich diese Investition bereits nach kurzer Zeit durch effektiveren Softwareeinsatz.

In größeren Verwaltungseinheiten kann diese Ressource Know-How so weit anwachsen, dass sich die Eigenentwicklungen vor den Lösungen der kommerziellen Softwarehersteller (egal ob OS/FS oder proprietär) nicht mehr zu verstecken brauchen. Das kann so weit gehen, dass die eigene Lösung die mit Abstand rentablere Variante ist. Das ist eine Entwicklung, die sich verstärken wird, je spezieller die Software wird.

4.8. Geteiltes Wissen ist einfach multiplizierbar, da die öffentlichen Verwaltungen bereits stark vernetzt sind

Anders als bei der materiellen Warenwirtschaft ist Wissen praktisch verlust- und kostenfrei multiplizierbar. Wenn eine physikalische Ware getauscht wird (z.B. ein Fahrzeug gegen eine Geldsumme), dann wechselt ein Gegenstand den Besitzer, das heißt der Verkäufer hat nach dem Verkauf keinen Zugriff mehr auf die Ware. Bei der "Ware" Wissen ist das anders, durch das Weitergeben (z.B. auch Verkaufen) verliert der Verkäufer das Wissen nicht, der Transfer gleicht einer Verdoppelung oder Multiplikation. Damit sollte schnell klar werden, dass der traditionelle Warenverkehr mit dem Gut Wissen nicht wirklich kompatibel sein kann.

Das reflektiert sich auch in den Lizenzbedingungen der Hersteller proprietärer Software. Sie gehen sogar noch weiter als beim physikalischen Warenverkehr – es wird lediglich eine temporäre Nutzerlizenz gestattet, die unter bestimmten Bedingungen auch wieder entzogen werden kann.

Source Code ist nichts anderes als kodiertes (in einer Programmiersprache strukturiertes) Wissen. Wird die obige Logik angewendet, so wird schnell klar, dass es keinen physikalischen Grund gibt, den Zugriff auf Software zu beschränken, da durch deren Nutzung dem Besitzer (oder Erfinder, Verkäufer) kein materieller Verlust entsteht. Gerade öffentliche Verwaltungen bieten sich deshalb als Multiplikatoren von Software an – Voraussetzung ist der allgemeine und uneingeschränkte Zugriff darauf.

5. Die Anwendung von OS/FS Konzepten als Lösung für Softwarebedarf in der öffentlichen Verwaltung

Die oben genannten Argumente zeigen, dass die Konzepte OS/FS für den Einsatz in der öV idealtypisch geeignet sind. Die Konzepte sind auch ohne zentrale (verpflichtende) Steuerung in Form von Verordnungen anwendbar, da sie problemorientiert sind und sich

evolutionär entwickeln. Der Vorteil ist die weitgehende Selbststeuerung der Entwicklung von Open Source Projekten. Reibungsverluste sind auch hier nicht unbekannt, für ein Problem kann es durchaus mehrere Softwarelösungen geben, allerdings wird sich langfristig die bessere Lösung durchsetzen, was nicht zuletzt auch wieder der Qualität der Software zugute kommt.

5.1. Ausnahmen und Sondersituationen

Für spezielle Lösungen und Nischenprodukte sind OS/FS Konzepte am wenigsten geeignet, da eine kleine Anwendergemeinschaft nicht die erforderliche Leistung erbringen kann, um das Produkt weiterzuentwickeln. Hier werden sich proprietäre Lösungen am längsten halten und auch Sinn machen.

5.2. Zusammenfassung

Die Techniken und Praktiken der OS/FS Gemeinschaft finden immer mehr Verbreitung und werden inzwischen auch verstärkt von proprietären Herstellern übernommen, angefangen von Userlisten und Foren bis hin zum Einsatz externer Betatester (die für das Testen zahlen müssen) und nicht zuletzt beim Endkunden, bei dem "die Banane engültig reift". Auch das zeigt, dass sich die derzeit stark abgegrenzten Bereiche proprietärer und Freier Software immer mehr annähern werden, da sich auch hier langfristig schlicht die besseren Konzepte durchsetzen werden.

*** * ***

Standards und Schnittstellen - Beispiel OGC WMS

Von Arnulf Christl & Astrid Emde

1. Standards und Schnittstellen: Einführung in WMS

Für die meisten Fragestellungen im IT Bereich haben sich nach und nach Standards und Konventionen entwickelt. Texte werden z.B. als ASCII Zeichen hinterlegt und sind damit zwischen den Systemen austauschbar. Ein sehr wichtiger Schritt für die Entwicklung der IT war die Implementierung und der inzwischen praktisch flächendeckende Einsatz von SQL als Abfragesprache für Datenbanken. Doch auch bei SQL gibt es verschiedene Dialekte und wenn man "in" eine Datenbank hineinsieht funktioniert vieles doch wieder anders als der kleinste gemeinsame Nenner SQL.

Im Bereich der GIS Technologie hat sich über die Jahre SQL auch als das Standardwerkzeug für den Zugriff auf alphanumerische Daten herauskristallisiert. Geometrische Daten werden historisch bedingt in den meisten GIS Softwarepaketen proprietär gehalten, oft in "geheimen" Formaten. Für den Einsatz der zunächst monolithischen Systeme war es sehr praktisch, alle Daten nur in eigenen Formaten vorzuhalten. Einerseits konnte ein für den Einsatzzweck optimiertes Format entwickelt werden, andererseits wurden die Abhängigkeit der Kunden von den monolithischen Systemen zementiert.

Das kann ein außerordentliches Problem werden, weil geographische Daten eine hohe Lebenserwartung haben. Ein Wechsel zu anderer Software ist deswegen mit enormem Aufwand und oft auch mit hohen Verlusten an Funktionalität und sogar Daten verbunden.

Die rasante Entwicklung der Hardware und die Verbreitung von Netzwerktechnologie, aber auch die immer stärkere inhaltliche Vernetzung unterschiedlicher IT-Bereiche hat die technischen Grundlagen und den Bedarf geschaffen, um auch im GIS – Bereich Standards zu schaffen.

1.1. Die OGC WMS Spezifikation

Einer der am weitesten verbreiteten Standards ist der Web Map Service (WMS) der den Standard definiert, über den Kartenbilder angefordert werden können. Dieser Dienst wird von vielen GIS Softwarepaketen unterstützt und soll hier beschrieben werden. Wichtig ist dabei die Kapselung der Datenhaltung, Aufbereitung und Präsentation hinter dem Dienst. Es ist für die WMS Spezifikation irrelevant, wie aus georeferenzierten Daten Karten produziert werden und auch in welchem Ursprungsformat die Daten vorliegen. Es ist lediglich erforderlich, dass der Dienst standardisierte Ergebnisse für standardisierte Anforderungen liefert. Die WMS Spezifikation definiert drei Aufrufe:

- getCapabilities
- getMap

- getFeatureInfo (optional)

Diese drei parametrierten Aufrufe reichen aus, um die Diensteigenschaften anzufordern, mit den darin abgelegten Daten eine Karte mit einer Auswahl an Themenebenen anzufordern und Informationen zu einzelnen Objekten abzufragen.

Im normalen Betrieb sollte die WMS Schnittstelle so effektiv durch Software gekapselt werden, dass die Nutzer gar nicht merken, dass sie eine WMS-Anfrage gestellt haben. Für Entscheider ist es wichtig, die WMS Schnittstellendefinition als unumgänglichen Standard beim Aufbau von komplexen GI-Systemen zu berücksichtigen. Geo-Administratoren, System-Architekten und vor allem Entwickler sollten dagegen ein etwas tieferes Verständnis der Spezifikation mitbringen, um sie möglichst effektiv nutzen zu können.

1.2. Standardisierungsprozesse und das OGC

Standardisierungsbestrebungen für GIS-Fragestellungen gibt es so lange wie es GIS-Fragestellungen gibt. Auf der Webseite des OpenGIS Consortium[6] findet sich ein kurzer historischer Abriss:

http://www.opengis.org/about/?page=history

Die ersten Bestrebungen werden auf das Jahr 1980 zurückführt. Seitdem haben sich die unterschiedlichsten Prozesse parallel und teilweise redundant mit den verschiedensten Interessen dezentral entwickelt. Einige dieser Prozesse wurden untereinander koordiniert und zusammengeschlossen. Eine ernstzunehmende Mehrheit rief zunächst die "GRASS-GIS Users Organization" ins Leben (1991) aus der die Institution OGF (OpenGIS Foundation) hervorging (1993).

Der direkte Vorläufer des OGC, das OGF (eine Stiftung) firmierte zunächst als "501 (c)3 not-for-profit charitable foundation", änderte seinen Status dann in "501 (c)6 not-for-profit trade association", mit der Begründung nur so den Gesetzen des Marktes gerecht werden zu können. Dieser Zusammenschluss firmierte unter dem Namen "OGIS Ltd.", der am 22. Oktober 1994 in "Open GIS Consortium, Inc." geändert wurde.

Das Consortium koordiniert seitdem eine Menge vielbeachteter Standards und Prozesse, und steht Herstellern, Wissenschaft und Forschung, aber auch Anwendern offen. Wenn wir jetzt sagen: "...auch in dieser Reihenfolge", dann ist das als ein klare Wertung zu verstehen.

Das OpenGIS Consortium ist Aufgrund politischer Konflitke zwischen einigen großen Herstellern in die Diskussion gekommen und hat dabei auch einen Imageverlust erlitten. Weitere Details dazu finden sich hier:

http://www.directionsmag.com/article.php?article_id=455

6 Zum Open GIS Consortium, siehe weiter unten (Kapitel 2)

Die Frage ist nun, wie ein Praxishandbuch zum Einsatz Freier Software und Open Source Konzepte mit diesen politischen Diskussionen umgeht. Wir machen es uns an dieser Stelle einfach und vermeiden weitere polarisierende Meinungen. Wir geben zunächst eine kurze Zusammenfassung über die historische Entwicklung des OpenGIS Consortium (OGC), um dann auf die praktische Anwendbarkeit lediglich eines Standards – die WMS Spezifikation – einzugehen.

1.3. Das OpenGIS Consortium – OGC

Das Open GIS Consortium (OGC) ist ein internationaler Zusammenschluss von Vertretern aus dem gewerblichen, öffentlichen und universitären Bereich. Das OGC wurde 1994 von 8 Mitgliedern gegründet. Im Laufe der Jahre ist die Mitgliederzahl bis Anfang 2004 auf 255 angewachsen. Die Mitglieder können je nach Status an den Spezifikationen mitarbeiten, lediglich beraten oder nur Einblick in die Entwicklung bekommen.

Ziel des OGC ist es, geographische Informationen und Dienstleistungen netz-, programm- und plattformübergreifend zugänglich zu machen. Durch spezifizierte Schnittstellen soll der reibungslose Austausch von Geodaten und der Zugriff auf standardisierte Dienste zwischen verschiedenen Systemen ermöglicht werden.

Das OGC definiert zu diesem Zweck eine Vielzahl von Spezifikationen, unter anderem ein Datenformat in XML Notation (GML), die Geometrieformate WKT (Well Known Texts) und WKB (Well Known Binaries), aber auch Dienstschnittstellen wie WMS (Web Map Service), WFS (Web Feature Service), SLD (Styled Layer Descriptor), WCS (Web Coverage Service), einen Katalogdienst für den strukturierten Zugriff auf die Dienste und vieles mehr.

Auf der OGC Homepage (http://www.opengis.org/) stehen die Spezifikationen, Schemata, Referenzmodelle und Discussion Papers zum Download zur Verfügung.

Die folgende Grafik gibt eine Übersicht über die OGC Web Services Architektur.

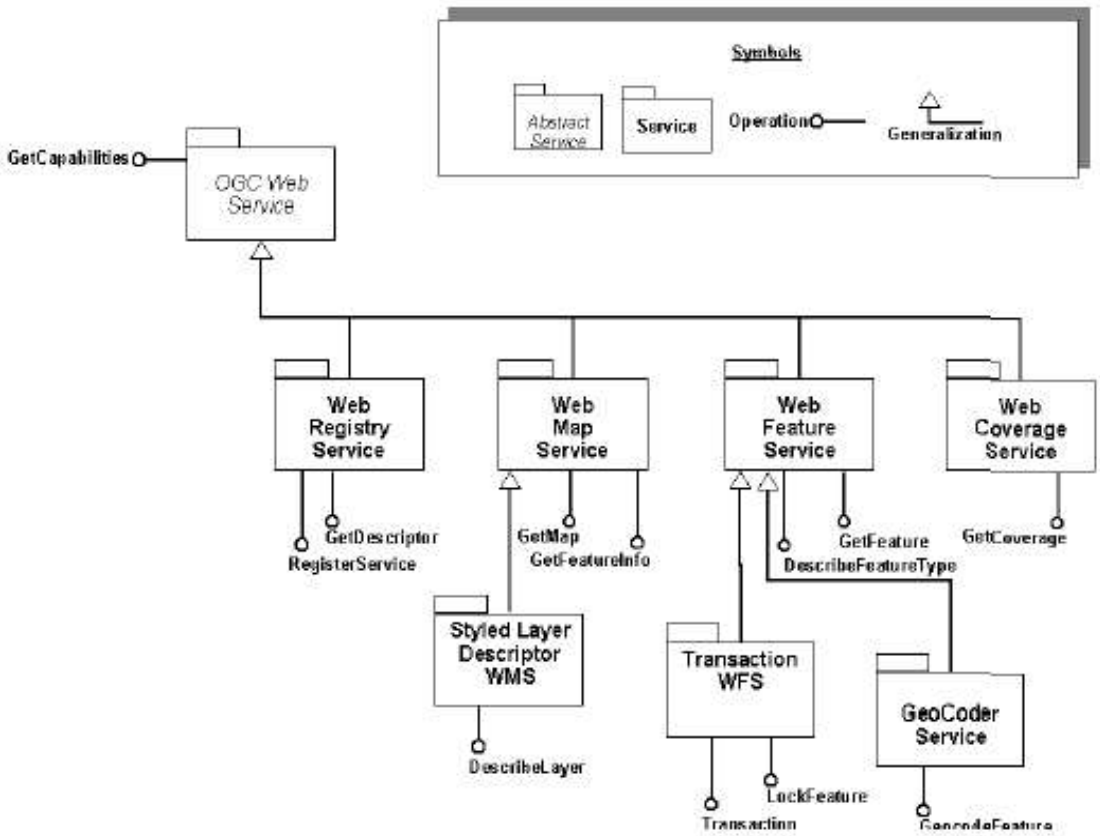

Abb 1.: OGC Web Services Architektur (aus: OGC 01-068r3 Web Map ServiceImplementation Specification, Fig 1)

Diese Abbildung zeigt lediglich einen kleinen Auschnitt der Dienste. Viele weitere Spezifikationen, die in den Arbeitsgruppen des OGC und deren Vorläuferinstitutionen entwickelt wurden, bilden die Basis eines standardisierten Rahmenwerkes zur Verarbeitung von GIS Daten. Für weitere Informationen verweisen wir auf die Online Referenzen des OGC.

2. Der Web Map Service (WMS)

Mit dem Web Map Service (WMS) wurde vom OGC ein Standard definiert, der sowohl die Syntax der Anfragen nach einem Kartenbild, als auch Format und Eigenschaften des Ergebnisses dieser Anfrage regelt. Von einem WMS (Dienst) werden nicht Geometriedaten angefordert, sondern deren visuelle Präsentation als Raster-Bild.

Die WMS-Spezifikation liegt mittlerweile in mehreren Versionen vor. Anhand der aktuellen Version WMS 1.1.1. soll der WMS im folgenden beschrieben werden (siehe OGC 01-068r3 Web Map Service Implementation Specification).

Ein WMS muss die folgenden 3 Anfragen beantworten können:

- getCapabilities
- getMap
- getFeatureInfo (optional)

Über einen Request auf einen dieser Aufrufe wird beim Service eine Anfrage gestellt. Der Service analysiert die Anfrage und liefert ein entsprechendes Ergebnis zurück.

- getCapabilities liefert ein XML Dokument mit den Eigenschaften des Services zurück,
- getMap liefert ein georeferenziertes Rasterbild mit Karteninhalten zurück,
- getFeatureInfo liefert die Ergebnismenge aller gefunde Objekte an einer Position zurück.

Die Parameter, mit denen ein WMS via URL angesprochen wird, sind standardisiert. Durch die Standardisierung der Requests können Anfragen an Kartenserver unterschiedlicher Anbieter gestellt werden, ohne auf proprietäre Eigenheiten eingehen zu müssen. Es erfolgt dabei eine klare Kapselung der Dienste und eine Trennung zwischen Anfrage, Datenhaltung und Aufbereitung. Nur so ist es möglich, heterogene Datenbestände, die in unterschiedlichen Systemen vorliegen, miteinander zu kombinieren und daraus neue Informationen abzuleiten und zu schaffen.

2.1. getCapabilities

Der getCapabilities-Aufruf gibt Metadaten zum WMS in Form eines XML-Dokuments zurück (XML - Extensible Markup Language). Der Aufbau des XML-Dokuments ist standardisiert und richtet sich nach dem Schema für den getCapabilities-Aufruf (WMS_MS_Capabilities.dtd). Die Metadaten umfassen allgemeine Informationen zum Service, Angaben zu den zur Verfügung stehenden Ebenen, Projektionssystemen, Koordinatenausschnitten und URLs zu den unterstützten WMS-Requests. Damit stellt das getCapabilities-Dokument alle erforderlichen Informationen bereit, um einen gültigen getMap und (sofern unterstützt) GetFeatureInfo-Request zu generieren.

2.2. getMap

Der getMap-Aufruf liefert vom WMS eine Karte als Rasterbild zurück. In dem Aufruf werden Angaben zum gewünschten Ausschnitt, dem Projektionssystem, den anzuzeigenden Layern, dem Bildformat sowie weiteren Parametern übergeben. Welche Parameterwerte – wie zum Beispiel die zur Verfügung stehenden Layer – für den Aufruf verwendet werden können, ist dem getCapabilities-Dokument zu entnehmen. Das Ergebnis ist ein georeferenziertes Rasterbild im gewünschten Bild-Format.

2.3. getFeatureInfo

Optional kann ein WMS auch den getFeatureInfo-Aufruf unterstützen, der alphanumerische Informationen zu Geometrieobjekten (Features) zurückliefert. Dabei werden der URL des getMap-Requests zusätzliche Parameter hinzugefügt. Es wird eine Punktgeometrie und die Namen der Layer, die abgefragt werden sollen, übergeben. Liegen auf den Layern an diesem Punkt Geometrieobjekte vor, werden Informationen zu diesen Objekten angezeigt. Die Rückgabe erfolgt in einem wählbaren gewünschten Format, meist als HTML-Seite.

3. WMS Unterstützung durch den UMN MapServer

Die University of Minnesota (UMN) ist Associate Member des OGC. Der UMN MapServer verfügt seit Version 3.5 über eine WMS kompatible Schnittstelle. Zur Einrichtung eines WMS mit dem UMN MapServer muss die zentrale Konfiguration, die MAP-Datei um METADATA-Parameter erweitert werden, die für die Generierung des getCapabilities-Aufruf benötigt werden. Anschließend kann das Projekt von Clients, die WMS kompatibel sind, angezeigt werden.

Der UMN MapServer ist in der aktuell verfügbaren stabilen Version (Anfang 2004 in der Version 4.x) nicht OGC WMS **konform** sondern lediglich **kompatibel**, da nicht alle Konformitätsvorschriften erfüllt werden, z.B. die nicht WMS-spezifische Behandlung von geometrisch verzerrten Aufrufen. Der UMN MapServer liefert immer unverzerrte Kartenbilder zurück. Die WMS Spezifikation fordert aber explizit auch das Zurückliefern verzerrter Kartenbilder. Ein Kartenbild, bei dem das Höhen-Breitenverhältnis der Bildgeometrie und der Koordinaten nicht übereinstimmen, wird vom UMN MapServer automatisch zentriert. Laut Spezifikation sollte das Kartenbild aber gestreckt werden, da nur exakt quadratische Pixel auf einem Monitor dargestellt werden können. Dies kann zu Verschiebungen bei der Überlagerung mit anderen Diensten führen, sollte im Normalfall aber durch sauber definierte Umgebungsrechtecke und passende Bildgrößen der Clientsoftware nicht auftreten.

3.1. WMS Schnittstellenspezifikation am Beispiel des UMN MapServer

Die technische Dokumentation des WMS Aufrufes entnehmen Sie bitte der OGC Web Map Service Implementation Specification (http://www.opengis.org/techno/specs.htm). Achten Sie auch darauf, dass die WMS Spezifikation in unterschiedlichen Versionen vorliegt, Anfang 2004 ist das die Version WMS 1.1.1.

3.1.1. Allgemeine Service Elemente

Bevor auf die drei Requests einzeln eingegangen wird, hier eine kurze Beschreibung der Regeln, die für alle drei Requests gültig sind.

Aufbau des HTTP-Requests

```
http://servername[:port]/pfad?parametername=parameterwert&parametername=...
```

In dem URL Aufruf haben neben den üblichen HTTP Einschränkungen (keine Sonderzeichen, Freizeichen, etc.) folgende Zeichen spezielle Aufgaben

Reservierte Zeichen	Beschreibung
?	Separator, der den Beginn der an den WMS übergebenen Parameter im URL-Aufruf anzeigt
&	Separator zwischen den einzelnen Parametern des WMS Aufrufes
=	Separator zwischen Parametername und –wert
/	Separator zwischen MIME type und subtype im FORMAT-Parameter
:	Separator Namespace und Identifier des SRS-Parameterwertes
,	Separator zwischen aufgelisteten Parameterwerten. Es dürfen keine Leerzeichen für die Abgrenzung verwendet werden.

Regeln für die Request-Parameter

Bei Parameternamen soll nicht zwischen Groß- und Kleinschreibung unterschieden werden, Parameterwerte können aber unterschieden werden. Die Reihenfolge, in der die Parameter im Request angegeben werden, ist beliebig.

Aufbau der Online Resource-URL für den HTTP Aufruf

Mit der Online Resource-URL wird der Web Map Service angesprochen. Bei der Online Resource-URL handelt es sich um einen URL Prefix. An die Online Resource URL werden Parameter angefügt, um einen gültigen Aufruf zusammenzustellen.

```
http://servername[:port]/pfad?{name[=wert]&
```

Der URL Prefix setzt sich aus dem Protokoll <`http://`>, dem Rechnernamen, der Portnummer (optional), dem Pfad zur WMS-Dienstkomponente (beim UMN MapServer die ausführbare Executable) und einem abschließenden `<?>` zusammen. Optional können serverspezifische Parameter übergeben werden, ein WMS sollte aber auch ohne diese in der Lage sein, einen gültigen Request zu beantworten.

Beispiel für eine Online Resource URL des UMN MapServer:

```
http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/germany.map&
```

3.1.2. Beispiel des getCapabilities-Aufrufs

Der Capabilities Aufruf wird als REQUEST Parameter an die Online Resource URL angehängt.

```
http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/germany.map&REQUEST=Capabilities&VERSION=1.0.0
```

In diesem Beispiel wird der Bestandteil `<map=d:/umn/germany.map>` zum Standard OGC WMS Aufruf hinzugefügt. Damit können über eine UMN MapServer URL (Serverdienst) auch unterschiedliche Karteninhalte in eigenen Diensten vorgehalten werden, indem einfach ein Verweis auf eine andere MAP-Datei gesetzt wird.

Der oben angegebene getCapabilities-Request liefert ein XML-Dokument zurück. Dieses soll im Folgenden näher beschrieben werden. Sie können dieses Beispiel Online aufrufen und testen.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE WMT_MS_Capabilities SYSTEM "http://www.digitalearth.gov/wmt/xml/capabilities_1_0_0.dtd" [
 <!ELEMENT VendorSpecificCapabilities EMPTY>
]>
<!-- end of DOCTYPE declaration -->
<WMT_MS_Capabilities version="1.0.0" updateSequence="0">
 <!-- MapServer version 4.0.1 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP OUTPUT=PDF OUTPUT=SWF
SUPPORTS=PROJ SUPPORTS=FREETYPE SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT SUPPORTS=WFS_SERVER
SUPPORTS=WFS_CLIENT INPUT=POSTGIS INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE -->
 <Service>
  <Name>GetMap</Name>
  <Title>Germany</Title>
  <Abstract>Germany</Abstract>
  <OnlineResource>http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/germany.map&</OnlineResource>
  <AccessConstraints>none</AccessConstraints>
 </Service>
 <Capability>
  <Request>
   <Map>
    <Format>
     <GIF/>
     <PNG/>
     <JPEG/>
     <WBMP/>
    </Format>
    <DCPType>
     <HTTP>
      <Get onlineResource="http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/germany.map&"/>
      <Post onlineResource="http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/germany.map&"/>
     </HTTP>
    </DCPType>
   </Map>
   <Capabilities>
    <Format>
     <WMS_XML/>
    </Format>
    <DCPType>
     <HTTP>
      <Get onlineResource="http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/germany.map&"/>
      <Post onlineResource="http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/germany.map&"/>
     </HTTP>
    </DCPType>
   </Capabilities>
   <FeatureInfo>
    <Format>
     <MIME/>
     <GML.1/>
    </Format>
    <DCPType>
     <HTTP>
      <Get onlineResource="http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/germany.map&"/>
```

```
        <Post onlineResource="http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/germany.map&"/>
       </HTTP>
      </DCPType>
    </FeatureInfo>
   </Request>
   <Exception>
    <Format>
     <BLANK/>
     <INIMAGE/>
     <WMS_XML/>
    </Format>
   </Exception>
   <VendorSpecificCapabilities/>
   <Layer>
    <Name>Germany</Name>
    <Title>Germany</Title>
    <SRS>epsg:31493</SRS>
    <LatLonBoundingBox minx="-0.582939" miny="46.7495" maxx="21.2066" maxy="55.3001"/>
    <BoundingBox SRS="EPSG:31493" minx="2.88661e+006" miny="5.23045e+006" maxx="4.28519e+006"
maxy="6.13001e+006"/>
    <ScaleHint min="0.000498903" max="4.98903e+013"/>
    <Layer queryable="0">
     <Name>Topographie</Name>
     <Title>Topographie</Title>
     <SRS>epsg:4326</SRS>
     <LatLonBoundingBox minx="5.86563" miny="47.2758" maxx="15" maxy="55.0556"/>
     <BoundingBox SRS="EPSG:4326" minx="5.86563" miny="47.2758" maxx="15" maxy="55.0556"/>
    </Layer>
    <Layer queryable="0">
     <Name>Grenze</Name>
     <Title>Grenze</Title>
     <SRS>epsg:4326</SRS>
     <LatLonBoundingBox minx="5.86563" miny="47.2758" maxx="15" maxy="55.0556"/>
     <BoundingBox SRS="EPSG:4326" minx="5.86563" miny="47.2758" maxx="15" maxy="55.0556"/>
     <ScaleHint min="0" max="4.98903e+013"/>
    </Layer>
    <Layer queryable="0">
     <Name>Bundeslaender</Name>
     <Title>Bundeslaender</Title>
     <SRS>epsg:4326</SRS>
     <LatLonBoundingBox minx="5.86417" miny="47.2747" maxx="15.0389" maxy="55.0567"/>
     <BoundingBox SRS="EPSG:4326" minx="5.86417" miny="47.2747" maxx="15.0389" maxy="55.0567"/>
     <ScaleHint min="0" max="449013"/>
    </Layer>
    <Layer queryable="0">
     <Name>Staedte</Name>
     <Title>Staedte</Title>
     <SRS>epsg:4326</SRS>
     <LatLonBoundingBox minx="6.05223" miny="47.4825" maxx="14.9981" maxy="54.9405"/>
     <BoundingBox SRS="EPSG:4326" minx="6.05223" miny="47.4825" maxx="14.9981" maxy="54.9405"/>
     <ScaleHint min="0" max="4490.13"/>
    </Layer>
    <Layer queryable="1">
     <Name>Postleitzahlbereiche</Name>
     <Title>Postleitzahlbereiche</Title>
     <SRS>epsg:4326</SRS>
     <LatLonBoundingBox minx="5.86629" miny="47.2736" maxx="15.0486" maxy="55.0583"/>
     <BoundingBox SRS="EPSG:4326" minx="5.86629" miny="47.2736" maxx="15.0486" maxy="55.0583"/>
    </Layer>
    <Layer queryable="0">
     <Name>Bahnlinien</Name>
     <Title>Bahnlinien</Title>
     <SRS>epsg:4326</SRS>
     <LatLonBoundingBox minx="5.98259" miny="47.4064" maxx="15" maxy="54.9045"/>
     <BoundingBox SRS="EPSG:4326" minx="5.98259" miny="47.4064" maxx="15" maxy="54.9045"/>
    </Layer>
    <Layer queryable="0">
     <Name>Fluesse</Name>
     <Title>Fluesse</Title>
     <SRS>epsg:4326</SRS>
     <LatLonBoundingBox minx="5.95192" miny="47.2864" maxx="14.9759" maxy="54.9107"/>
     <BoundingBox SRS="EPSG:4326" minx="5.95192" miny="47.2864" maxx="14.9759" maxy="54.9107"/>
     <ScaleHint min="0" max="623.629"/>
    </Layer>
    <Layer queryable="0">
     <Name>Strassen</Name>
     <Title>Strassen</Title>
```

```
      <SRS>epsg:4326</SRS>
      <LatLonBoundingBox minx="5.90508" miny="47.3136" maxx="14.9981" maxy="55.0157"/>
      <BoundingBox SRS="EPSG:4326" minx="5.90508" miny="47.3136" maxx="14.9981" maxy="55.0157"/>
    </Layer>
    <Layer queryable="1">
      <Name>Staedtepunkte</Name>
      <Title>Staedtepunkte</Title>
      <SRS>epsg:4326</SRS>
      <LatLonBoundingBox minx="6.01041" miny="47.4118" maxx="14.963" maxy="55.0175"/>
      <BoundingBox SRS="EPSG:4326" minx="6.01041" miny="47.4118" maxx="14.963" maxy="55.0175"/>
    </Layer>
    <Layer queryable="0">
      <Name>Postleitzahlbereichname</Name>
      <Title>Postleitzahlbereichname</Title>
      <SRS>epsg:4326</SRS>
      <LatLonBoundingBox minx="5.86629" miny="47.2736" maxx="15.0486" maxy="55.0583"/>
      <BoundingBox SRS="EPSG:4326" minx="5.86629" miny="47.2736" maxx="15.0486" maxy="55.0583"/>
      <ScaleHint min="0" max="174.616"/>
    </Layer>
    <Layer queryable="0">
      <Name>Staedtenamen</Name>
      <Title>Staedtenamen</Title>
      <SRS>epsg:4326</SRS>
      <LatLonBoundingBox minx="6.01041" miny="47.4118" maxx="14.963" maxy="55.0175"/>
      <BoundingBox SRS="EPSG:4326" minx="6.01041" miny="47.4118" maxx="14.963" maxy="55.0175"/>
    </Layer>
    <Layer queryable="0">
      <Name>Bundeslaendernamen</Name>
      <Title>Bundeslaendernamen</Title>
      <SRS>epsg:4326</SRS>
      <LatLonBoundingBox minx="5.86417" miny="47.2747" maxx="15.0389" maxy="55.0567"/>
      <BoundingBox SRS="EPSG:4326" minx="5.86417" miny="47.2747" maxx="15.0389" maxy="55.0567"/>
    </Layer>
   </Layer>
  </Capability>
</WMT_MS_Capabilities>
```

Das <Service>-Element enthält Metadaten zum Web Map Service. Benötigte Angaben innerhalb dieses Elements sind <Name>, <Title> und <OnlineResource>. Optional können weitere Angaben gemacht werden (z. B. <KeywordList>, <ContactInformation>).

Der UMN MapServer generiert das Capabilities XML aus den Angaben im Metadatabereich der MAP-Datei. Die Angaben des Service-Elements werden aus den Metadaten im Abschnitt WEB der MAP-Datei aufgebaut.

Das <Capability>-Element enthält die Angaben der vom Server unterstützten Requests und alle Informationen zu den eingebundenen Daten-Ebenen. Alle Ebenen der Karte werden gemeinsam von einem <Layer>-Element umschlossen, das als root-Layer bezeichnet wird. Jede einzelne Ebene selbst wird ebenfalls von einem <Layer>-Element umschlossen. Das umschließende Layer vererbt seine Eigenschaften an die untergeordneten Layer. Deswegen müssen nicht für alle Unter-Ebenen alle Informationen mit angegeben werden. Wenn die übergeordnete Ebene bereits die Koordinaten des Umgebungsrechtecks definiert, müssen diese für die darunter liegenden Ebenen nicht mehr zwingend angegeben werden.

UMN MapServer MAP Konfigurationsdatei und OGC WMS Capabilities

Bei dem UMN MapServer erfolgt die Konfiguration eines Kartendienstes über die MAP-Datei. Sie ist letztendlich nur ein aus der Historie etwas anders formatiertes Capabilities Dokument. Das wird deutlich, wenn die Blöcke gegenübergestellt werden. In den

folgenden Blöcken werden die Layerangaben im Mapfile der Ausgabe im Capabilities XML gegenübergestellt.

Eintrag aus der UMN MapServer MAP-Datei:

```
LAYER
    NAME "Topographie"
    STATUS ON
    DATA "topo.shp"
    TYPE POLYGON
    PROJECTION
      "init=epsg:4326"
    END
    METADATA
      "WMS_SRS"    "epsg:4326"
      "WMS_TITLE"    "Topographie"
      "WMS_FEATURE_INFO_MIME_TYPE"    "text/html"
END..
```

Daraus generiert der UMN MapServer den folgenden Abschnitt in dem Capabilities Dokument:

```
<Layer queryable="0" opaque="0" cascaded="0">
<Name>topographie</Name>
      <Title>Topographie</Title>
      <SRS>epsg:4326</SRS>
      <LatLonBoundingBox minx="5.8" miny="47.2" maxx="15" maxy="55.6"/>
      <BoundingBox SRS="EPSG:4326" minx="5.8" miny="47.2" maxx="15" maxy="55.05"/>
</Layer>
```

Die Namen der Ebene werden direkt der MAP-Datei entnommen, die Koordinaten der BoundingBox (Umgebungsrechteck) werden bei einem Capabilities Request zur Laufzeit dynamisch vom UMN MapServer berechnet und zurückgeliefert.

Eigenschaften eines Layer Elementes „Capabilities“

<Title>

Titel (fakultativ)

<Name>

Nur Layer mit einer <Name>-Angabe können abgefragt werden (fakultativ)

<Abstract>

Beschreibung der Layer (optional, aber empfohlen)

<SRC>

Spatial Reference System (SRS) der Layer (fakultativ)

<LatLonBoundingBox>

Jede Ebene muss ein <LatLonBoundingBox>-Element beinhalten. Es kann direkt im

Layer-Tag angegeben werden, aber auch vom Parent-Layer geerbt werden. Die LatLonBoundingBox gibt das kleinste Umgebungsrechteck des Layers in EPSG:4326 (WGS 84, geographische Projektion) an.

<BoundingBox>

Jedes Layer hat Null oder mehrere <BoundingBox>-Elemente. Die BoundingBox gibt das Umgebungsrechteck einer Ebene für ein definiertes Projektionssystem an.

<ScaleHint>

Der Scalehint ist ein technisch richtiger aber unter praktischen Gesichtspunkten etwas glückloser Versuch, mit Maßstäben umzugehen. Die Theorie ist folgende: Die WMS Spezifikation definiert den ScaleHint als diagonale Länge in Realwelt-Metern des in der Mitte der Kartenbildes liegenden Pixels. Ein ScaleHint von 9 bedeutet also, dass ein Bildpixel 9 Meter Durchmesser hat. Ein Kartenbild mit 400 Pixeln Breite stellt also Wurzel aus 9 * 400 = 1200 Meter da. Je nach Projektion ist dieses Ergebnis natürlich mehr oder weniger verfälscht, weil der ScaleHint nur für den Mittelwert des Ausschnittes gilt. Im Detail nachzulesen in der WMS Spezifikation 1.1.0 Abschnitt 7.1.5.4.

Das nächste Release der WMS Spezifikation soll zwei neue Parameter enthalten, den MinScaleDenominator und MaxScaleDenominator. Diese werden in der SLD 1.0 Spezifikation bereits verwendet und werden die Arbeit vereinfachen.

Innerhalb des Layer-Elements können noch weitere optionale Parameter angegeben werden. Neben den beschriebenen Elementen kann ein <Layer> mehrere optionale Attribute enthalten, z.B: queryable, cascaded, opaque, noSubsets, fixedWidth, fixedHeight. Es handelt sich dabei um optionale Angaben, wobei der default-Wert jeweils 0 ist.

<queryable>

Der Tag <queryable> [0, 1]entscheidet darüber, ob der WMS den getFeatureInfo Request für eine Ebene unterstützt. Der UMN MapServer gibt für jeden Layer, für den in der MAP-Datei ein Abfrage-Template angegeben wurde, im Capabilities Dokument queryable="1" aus.

3.1.3. Beispiel eines getMap-Aufrufs

Der getMap-Request fordert beim WMS eine Karte an. Diese wird als Rasterbild zurückgeliefert. Im folgenden werden die notwendigen Parameter des getMap-Aufrufs näher betrachtet.

Wird der unten aufgeführte GetMap-Aufruf abgeschickt, so liefert der WMS ein Rasterbild zurück (Abbildung getMap), in dem die Ebenen Staedte, Postleitzahlbereiche, Fluesse, Bahnlinien, Strassen, Staedtepunkte, Postleitzahlbereichname, Staedtenamen abgebildet werden (siehe Abbildung unten).

http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/germany.map&WMTVER=1.0.0&REQUEST=map&LAYERS=Staedte,Postleitzahlbereiche,Fluesse,Bahnlinien,Strassen,Staedtep

unkte,Postleitzahlbereichname,Staedtenamen&STYLES=default,default,default,default,default,default,default,default,default,default,default,default,default&SRS=EPSG:31493&BBOX=3352353.809748938,5614614.608169725,3370504.3611455816,5627447.163615758&WIDTH=314&HEIGHT=222&FORMAT=PNG&BGCOLOR=0xffffff&TRANSPARENT=TRUE&EXCEPTIONS=INIMAGE

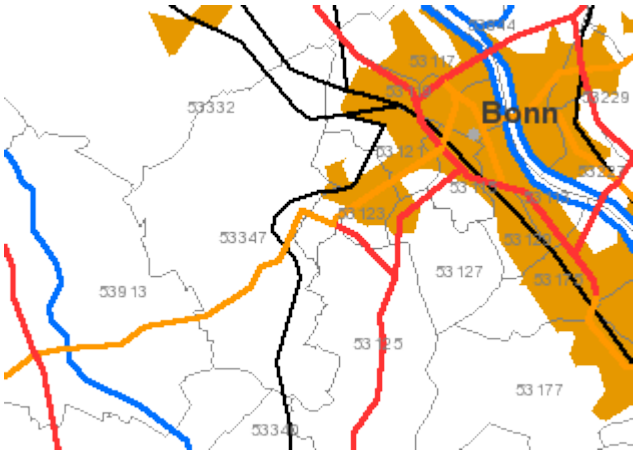

Abbildung GetMap: Das Ergebnis eines GetMap Aufrufes ist ein Rasterbild.

Bei diesem Beispiel lässt sich auch die Größe des Ausschnittes in der Realwelt nachrechnen. Bei einer Ausdehnung von Koordinate Rechtswert 3363162.2716981005 bis 3369325.54833812 ergeben sich 6163,2766400195 Meter (man beachte die vielen, wichtigen Nachkommstellen ;-)

GetMap Parameter

Parameter	Beschreibung
VERSION = 1.1.1	In der Spezifikation 1.0.0 heißt dieser Parameter noch WMTVER
REQUEST = [getMap]	Angabe des OGC WMS Abfragetyps
LAYERS = [Liste mit Ebenen]	Komma-separierte Auflistung der Ebenen aus dem Capabilities Dokument
STYLES = [default, SLD Dokument]	Darstellungsvorschrift der Layer in einem SLD Dokument (fakultativ). Wenn nur ein Style vorliegt, ist das automatisch der Style <default>.
SRS = [EPSG:xxxx, AUTO:xxxx]	Koordinatenbezugssystem (Spatial Reference System) entweder aus dem EPSG Namespace oder AUTO definiert.
BBOX=[minx,miny,maxx,maxy]	Koordinaten des Umgebungsrechtecks des gewünschten Ausschnitts in den Einheiten des Projektionssystems.

Parameter	Beschreibung
WIDTH = [integer]	Breite des Bildes in Pixeln
HEIGHT= [integer]	Höhe des Bildes in Pixeln
FORMAT = [output_format]	Die Notation erfolgt durch Voranstellung des Typs gefolgt von einem Schrägstrich und dem Format, z.B.: `image/png`
TRANSPARENT = [TRUE, FALSE]	Kartenhintergrund-Transparenz. Nur bei transparenten Formaten, z.B. PNG und GIF
BGCOLOR = [color_value]	Hintergrundfarbe der Karte, (optional) Angabe im Format 0xRRGGBB, default BGCOLOR=0xFFFFFF (weiß)
EXCEPTIONS = [exeption_format]	Ausgabeformat für Fehlermeldungen, default = application/vnd.ogc.se_xml

Weitere optionale Parameter des GetMap-Requests werden in der OGC 01-068r3 Web Map Service Implementation Specification [3] beschrieben.

Werden beim getMap-Aufruf keine Parameter übergeben, wird die gesamte Karte mit allen Layern zurückgegeben.

3.1.4. Beispiel eines GetFeatureInfo-Aufrufs

Die Unterstützung des getFeatureInfo_requests durch einen WMS ist optional. Über den GetFeatureInfo-Aufruf können weitergehende Informationen zu auf der Karte dargestellten Geo-Objekten (Features) abgefragt werden. Der Aufruf ist nur für Layer gültig, die über das Attribut queryable="1" verfügen.

Beispiel für einen GetFeatureInfo-Request:

http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/germany.map&WMTVER=1.0.0&REQUEST=feature_info&SRS=EPSG:31493&BBOX=3361111,5620655,3370186,5627071&WIDTH=314&HEIGHT=222&FEATURE_COUNT=30&QUERY_LAYERS=Postleitzahlbereiche&X=181&Y=105&INFO_FORMAT=text/html

GetFeatureInfo Parameter

Parameter	**Beschreibung**
VERSION = 1.1.1	In der Spezifikation 1.0.0 heißt dieser Parameter noch WMTVER
REQUEST = [getFeatureInfo]	Angabe des OGC WMS Abfragetyps

Parameter	Beschreibung
SRS = [EPSG:xxxx, AUTO:xxxx]	Koordinatenbezugssystem (Spatial Reference System) entweder aus dem EPSG Namespace oder AUTO definiert.
BBOX=[minx,miny,maxx,maxy]	Koordinaten des Umgebungsrechtecks des gewünschten Ausschnitts in den Einheiten des Projektionssystems.
WIDTH = [integer]	Breite des Bildes in Pixeln
HEIGHT= [integer]	Höhe des Bildes in Pixeln
FEATURE_COUNT = [integer]	Maximale Anzahl zurückgelieferter Ergebnisdaten-sätze. Voreinstellung: FEATURE_COUNT=1
QUERY_LAYERS = [Liste mit Ebenen]	Komma-separierte Auflistung der Ebenen aus dem Capabilities Dokument, die abgefragt werden sollen.
X = [integer]	Pixelspalte im Kartenbild. Der Punkt liegt innerhalb der Grenzen der WIDTH- und HEIGHT Parameter. Dabei liegt der Ursprung in der oberen linken Ecke.
Y = [integer]	
INFO_FORMAT = [text/html]	Ausgabeformat für die Ergebnismenge. MIME Type z.B. INFO_FORMAT = PNG, INFO_FORMAT = text/html, INFO_FORMAT = GML)

Neben diesen Pflichtwerten werden oft auch die restlichen Parameter eines getMap Requests mit übergeben. Daraus kann eine Ergebniskarte abgeleitet und angezeigt werden. Das Abfrageergebnis wird in dem angegebenen Format zurückgeliefert, vorausgesetzt der Dienst kann dieses Format generieren. Die Rückgabe der Ergebnisse unterliegt keinen weiteren Einschränkungen, es können also auch zusätzliche Informationen geliefert werden.

getFeatureInfo Request im UMN MapServer

Der UMN MapServer gibt voreingestellt HTML-Dokumente zurück. Der Aufbau dieser HTML-Seite kann in Abfrage-Templates vorbereitet werden, auf die ein Eintrag in der MAP-Datei verweist. Zusätzlich bietet der UMN MapServer auch an, einen Karten-ausschnitt mitzuliefern, in dem das Ergebnis hervorgehoben dargestellt wird (Abbildung Kartenausschnitt). Diese Funktionalität ist nicht in der WMS Spezifikation vorgeschrieben, kann aber als Erweiterung in dem Ergebnis enthalten sein.

Postleitzahlbereich

Postleitzahl	Stadt
53111	Bonn

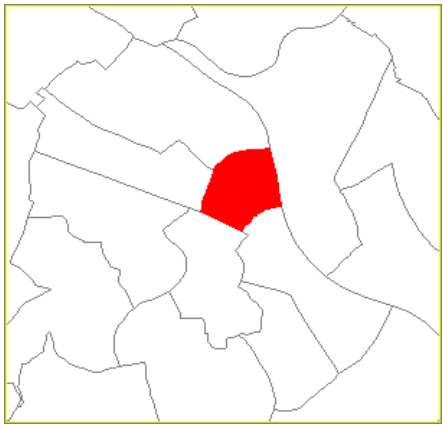

Abbildung Kartenausschnitt: Ergebnis eines FeatureInfo-Requests an den UMN MapServer.

Links

[1] Open GIS Consortium (OGC) Homepage:

http://www.opengis.org/

[2] Downloadbereich der OGC Homepage

http://www.opengis.org/techno/specs.htm

[3] Open GIS Consortium (OGC) WMS Implementation Specification:

http://www.opengis.org/docs/01-068r2.pdf

* * *

Einführung in WebGIS und in die Funktionsweise des UMN MapServer

Von Till Adams

1. WebGIS

GIS-Applikationen im Inter- oder Intranet - kurz WebGIS - machen Informationen für eine große Zahl von Anwendern zugänglich, sind plattformunabhängig und erfordern keine Installation proprietärer und kostenintensiver Desktop-GIS-Software. Zur Anzeige der GIS-Anwendungen ist lediglich ein Internet-Browser erforderlich. Schon diese Aspekte verdeutlichen das immense Potential von WebGIS-Anwendungen.

Generell scheinen die Zeiten, in denen bei der Einführung eines GI-Systems in einer Verwaltung erstmal ein *paar handvoll* Desktop-GIS-Lizenzen gekauft wurden, endgültig vorbei. Dennoch ist für viele Nutzer, besonders auch in öffentlichen Verwaltungen, ein kartenbasierendes Auskunftssystem sehr interessant.

Mit zunehmender Geschwindigkeit und Entwicklung der Netzwerk-Technik wird das Auslagern von GIS-Funktionalität in den normalen Webbrowser ohne lokale Installation und Lizenzkosten immer interessanter. Dazu kommt die Weiterentwicklung von Geodatenbanken, wie z.B. PostGIS, die zunehmend erlauben, echte GIS-Funktionalität wie z.B. das Puffern von Objekten oder das Editieren in den WebClient zu verlegen.

WebGIS bedeutet letzten Endes, dass der Nutzer zumindest eine dynamische Karte sieht, sich in dieser frei bewegen kann und diese selbst, aufgrund einer bestimmten Auswahl an Themen, frei gestalten kann. Dies bedeutet generell, dass eine serverseitige Architektur aufgebaut werden muss. Diese sieht im einfachsten Fall so aus, dass ein Webserver, z.B. Apache, und ein Kartenserver, z.B. der UMN MapServer, aufgesetzt werden und ihr Zusammenspiel erlaubt, dass der Nutzer an einer Benutzeroberfläche – dem Client – navigiert, diese Anfrage an den MapServer geschickt wird, dieser eine neue Karte produziert und diese an den Client zurückschickt (s. Abbildung 1).

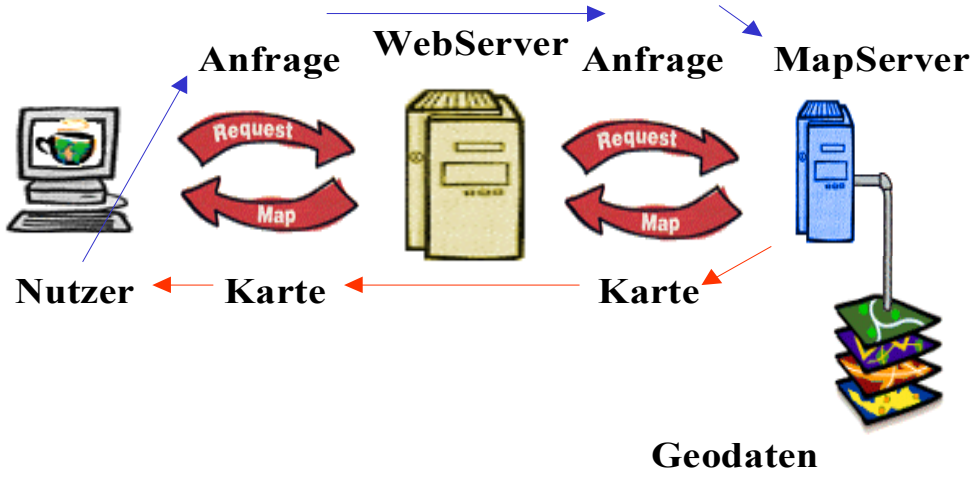

Abbildung 1: einfache WebGIS Client Server-Architektur

2. Der UMN MapServer

Der UMN MapServer arbeitet als sogenanntes CGI-Script (Common Gateway Interface) in der Script-Schnittstelle des WebServers. Der Benutzer am Browser hat die Berechtigung, Programme, die sich in diesem Verzeichnis befinden, auf dem Server auszuführen. Das MapServer CGI-Script benötigt von hier aus Zugriff auf weitere Dateien. Diese werden im Folgenden besprochen.

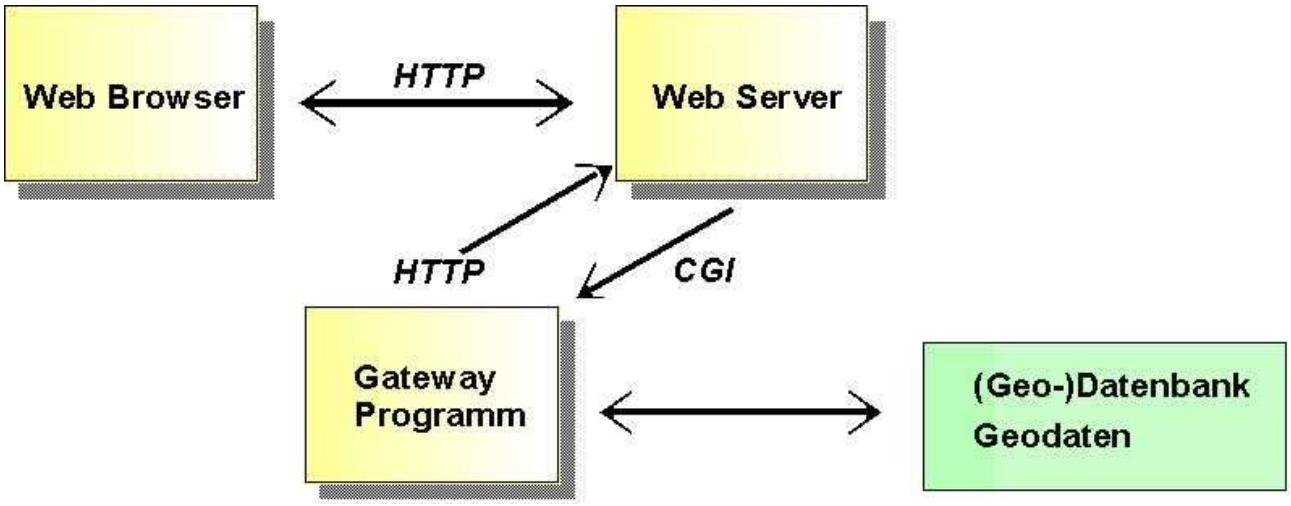

Abbildung 2: Funktionalität des UMN MapServers

Der Nutzer am Webbrowser kommuniziert über das HTTP-Protokoll mit dem Webserver,

soweit sind wir noch in der normalen Funktionalität des Internets allgemein. Eine Kartenanfrage, also z.B. „Zoom in die Karte“, wird vom Webserver an den UMN MapServer weitergeleitet. Dies geschieht über die CGI-Schnittstelle zwischen Webserver und CGI-Script. Der MapServer erhält nun eine Reihe von Parametern und verarbeitet diese. Dabei greift er über eine Netzverbindung oder lokal auf Geodaten zu und produziert eine neue Karte in Form eines Png- oder Gif-Bildes. Dieses wird via des HTTP Protokolls wieder an den Webserver und von da an den Webbrowser zurückgeliefert (s. Abb. 2).

Die Steuerung der eigentlichen Kartenanfrage wird über eine Projektdatei oder kurz „die MAP-Datei“ ("*.map") gesteuert. Die MAP-Datei ist in etwa vergleichbar mit einer normalen Desktop-GIS Projektdatei. Bei jeder Anfrage des Nutzers werden für Karte, Übersichtskarte, Legende und Maßstabsleiste temporäre Bilddateien erzeugt, die an entsprechenden Stellen eines HTML-Templates - des sogenannten "Clients" - eingesetzt werden. Bei jeder Anfrage durch den Benutzer, werden entsprechend der Anfrage neue temporäre Bilder angelegt und auch angezeigt. Das Resultat ist eine völlig dynamische Karte im Internet.

Datengrundlage

Datengrundlagen können im Netz verteilte Daten sein. Der UMN MapServer hat eine direkte Schnittstelle zu diversen Rasterformaten und zum Shape-File. Mit der GDAL und OGR-Bibliothek lassen sich aber auch andere Vektordatenformate wie GML und WFS oder auch MapInfo TAB Dateien direkt anzeigen.

Eine interessante Kombination ist die Verwendung des UMN Mapservers in Kombination mit einer Datenbank, in der auch die Geodaten vorgehalten werden können. Als Beispiel mag hier die PostgreSQL-Erweiterung PostGIS dienen. Das Speichern der Geometrien in der Datenbank bietet vor allem enorme Performance-Vorteile gegenüber einer Ablage in einem Dateisystem.

Wo innerhalb Ihres Netzwerks die Daten gespeichert sind, ist eigentlich egal, Hautpsache ist, man kann einen Pfad (relativ oder absolut) von der MAP-Datei zur entsprechenden Datei herstellen.

Die MAP-Datei

Die zentrale „Schaltstelle“ einer UMN MapServer Anwendung ist die MAP-Datei. Diese besteht aus Blöcken, die jeweils durch ein Schlüsselwort gestartet und ein "END" beendet werden. Einzige Ausnahme bildet der Header, also der oberste Abschnitt der Projektdatei, der ohne Schlüsselwort anfängt:

Die Dateien, die hier und im Folgenden besprochen werden, finden Sie auf der AveiN! CD im Ordner „Anwendungen/theAveiNDemo“ (http://www.terrestris.de/links/downloads.html). Die MAP-Datei heißt „marokko.map“, die zugehörige Benutzeroberfläche „Marokko.html“. Sie können die freie Software HTML-Kit, die sich auch auf der CD befindet, zum Anschauen und Bearbeiten der Dateien verwenden. HTML-Kit (ein Open Source Editor) bietet Ihnen eine Zeilennummerierung, auf die hier Bezug genommen wird.

```
NAME 'AveinDemo'
SIZE 450 300                              # Pixelgrösse des Bildes
EXTENT -13.777 27.249 -0.383 36.328       # Min X/Y/max X/Y
UNITS meters                              # Einheiten
SHAPEPATH 'data/'                         # relativer Datenpfad zur Datenquelle
SYMBOLSET 'symbols/symbset'               # Pfad zur Symboldatei
FONTSET 'fonts/fonts.fnt'                 # Pfad zur Fontdatei

...
  WEB
  ...
  END # Ende Websektion
END # Ende Header
```

Der Header der MAP-Datei fängt mit dem Parameter NAME an. Nach der Websektion werden beide Tools durch "END" geschlossen.

Der Header enthält für das gesamte Projekt gültige Angaben, die letztendlich auch das Aussehen Ihrer Karte bestimmen. Die Websektion definiert das Verhalten der Applikation und kann auch für OGC-konforme WebMap Services (WMS) erforderliche Metadaten enthalten. Weiterhin wird hier das HTML-Template (=Clientoberfläche) definiert. Es wird also angegeben, welcher Client mit dieser MAP-Datei zusammenarbeiten soll.

Referenzkarte

Die Referenz-Sektion definiert die Referenzkarte, also die kleine Übersichtskarte Ihrer Anwendung:

```
REFERENCE
  STATUS ON                                  # Karte wird produziert
  IMAGE graphics/ref_karte.png               # Bild, das verwendet wird
  SIZE 160 80                                # Pixelgrösse in x/y
  EXTENT -13.777 27.249 -0.383 36.328        # s.o.
 ...
END  #REFERENCE
```

Legende

Die Legenden-Sektion enthält eine eigene Labelsektion, die die Beschriftungsfarbe der Objekte in der Legende festlegt:

```
LEGEND
  STATUS ON
  KEYSIZE 16 8                                  # Größe der Legendensymbole
  TEMPLATE 'legend.html'                        # Legendentemplate
  LABEL COLOR 120 120 120                       # Farbe der Beschriftung
  END                                           # ENDE LABEL
END                                             #LEGEND
```

Dabei wird auf eine weitere HTML-Seite – das sogenannte Legendentemplate verwiesen. Dieses befindet sich im gleichen Ordner, in dem auch die MAP-Datei abgespeichert ist und heißt „legend.html“. Der MapServer ist in der Lage, diese Datei zu interpretieren. Im Quelltext der Seite legend.html steht folgendes:

```
[leg_group_html opt_flag=1] <input type="checkbox" name="layer" value="[leg_group_name]" [[leg_group_name]_check] title="Thema ein/ausschalten" onclick="neuladen();"> [leg_group_name]<br>[/leg_group_html]

[leg_class_html opt_flag=1]  <img src="[leg_icon width=16 height=8]" align="center">  [leg_class_name]<br>[/leg_class_html]
```

Generell gibt es einen öffnenden Tag und einen schließenden – alles was dazwischen steht, wird vom MapServer interpretiert, alles andere ignoriert. Alle eckigen Klammern symbolisieren, dass hier Inhalt dynamisch vom MapServer eingefügt wird.

Der Tag [leg_group_html] öffnet eine Gruppe – man spricht hier also die Layer über ihre Gruppe, nicht über ihren Namen an. Als nächstes wird ein HTML-Formularelement

definiert, eine „Checkbox“ - das ist die Box, mit der Sie den Layer hinterher an- und ausschalten. Gleichzeitig wird diese Checkbox mit einer JavaScript-Funktion belegt, die besagt, dass beim Anklicken dieser Checkbox, die Funktion „neuladen()“ ausgeführt werden soll. Der schließende Tag [/leg_group_html] beendet diesen Teil. Man hat nun eine Checkbox und einen Gruppennamen, der vom MapServer dynamisch eingeführt wird.

Im nächsten Tag „[leg_class_html]“ werden die einzelnen Klassen dargestellt:

Es wird eine Bildquelle (img src=[leg_icon] ...) definiert, die der MapServer dynamisch ausfüllt. Beachten Sie die „width“ und „height“ Definitionen, die hier die „Keysize“, die in der MAP-Datei definiert wird, „übertünchen“. Anschließend wird in [leg_class_name] noch der Name der Klasse eingesetzt.

Die Maßstabsleiste

```
#
# Start of scalebar
#
SCALEBAR
 STATUS ON                          # Maßstabsleiste anzeigen?
 STYLE 0                            # Alternativ existiert noch der STYLE 1
 INTERVALS 4                        # 4 Unterteilungen
 IMAGECOLOR 255 255 255             # Hintergrundfarbe weiß (Farben als RGB)
 LABEL                              # Beschriftung
  COLOR 0 0 0                       # Beschriftungsfarbe schwarz
  SIZE SMALL                        # kleine Beschriftung
END  #ENDE LABEL
 SIZE 200 3                         # Größe: 200 Pixel in x und 3 in y Richtung
 COLOR 255 255 0                    # Farbe der Scalebar gelb
 BACKGROUNDCOLOR 55 55 55           # und schwarz
 OUTLINECOLOR 100 100 100           # Umrissfarbe grau
 UNITS KILOMETERS                   # Einheiten (m/km/feet/miles)
END   #SCALEBAR
```

Symbole

In unserer Beispieldatei folgen danach einige Beispielsymbole, die definiert wurden. Es gibt viele Möglichkeiten, Objekte im UMN MapServer zu symbolisieren, wobei hier nur wenige der Möglichkeiten gezeigt werden können. Ein Beispiel für einen einfachen Punkt:

```
SYMBOL
 NAME 'circle'
 TYPE ELLIPSE
```

```
 FILLED TRUE
 POINTS 1 1 END                  # in ein gedachtes Koordinatensystem wird ein Punkt (1|1)
END                              # gesetzt
```

Themendefinitionen

Nun folgen die einzelnen Layer-Definitionen, wobei hier für jede Klasse eigene "CLASS" Objekte angelegt werden müssen. Grundsätzlich muss dabei noch zwischen Raster- und Vektordaten, und dabei weiterhin zwischen Polygonen, Linien und Punkten unterschieden werden.

Die Themendefinitionen fangen immer mit „LAYER“ an und werden mit einem „END # END OF LAYERFILE“ geschlossen.

Grundsätzlich gilt, was in der MAP-Datei oben definiert wird, wird als erstes angezeigt!

Daher wird zunächst das Hintergrundbild als Rasterbild definiert:

```
LAYER
  NAME 'Maroc'                                                    # Name des Themas
  GROUP 'Maroc'                                                   # Thema gehört zur Gruppe
  TYPE RASTER                                                     # Bildquelle, also z.B: Tiff
  DATA 'c:/usr3/htdocs/mapserver/theAveinDemo/data/maroc2.tif'
  # Pfad zur Datenquelle (geht auch relativ)
  STATUS ON                                                       # Bild initial an
# Stellen Sie den Status einmal auf DEFAULT und versuchen Sie das Thema auszuschalten
END # end of layer file
```

Und nun werden die einzelnen Vektorthemen definiert:

```
LAYER
  NAME agric
  DATA agric          # Die MAP-Datei liegt unter c:/usr3/htdocs/mapserver/theAveinDemo/
                      # von hier ergibt der im Header definierte SHAPEPATH "data/"
                      # dass die Daten unter
                      # c:/usr3/htdocs/mapserver/theAveinDemo/ + Shapepath liegen
                      # und agric heißt: Achtung, niemals die Endung „.shp" mitangeben
  STATUS ON
  TYPE Polygon              # Polygonshape
  CLASSITEM 'Danger'        # Spalte der Attributtabelle auf der klassifiziert wird

 CLASS
  NAME agriculture
```

```
  EXPRESSION /^9999xxx/                # nimm alles was mit 9999xxx anfängt
 END

 CLASS
  NAME 'Low'
  EXPRESSION /^Low/
  COLOR 255 255 0                 # Zeichne diese Klasse in gelb
  OUTLINECOLOR 0 0 0              # mit schwarzem Umriß
END  # CLASS
 ...

END # LAYER FILE
```

Die MAP-Datei wird beendet durch ein

```
END  # MAPFILE
```

Templates

Eine einfache Benutzeroberfläche / der HTML Client

Im Header der MAP-Datei wird der Client, ein sogenanntes „Template“ definiert, welches als eine Art „Lochmaske“ fungiert. Das Template ist eigentlich nichts anderes, als ein HTML-Formular, in das an entsprechenden Stellen die Elemente der WebGIS Anwendung platziert werden. Bei jedem Klick auf der Karte werden alle Parameter, die in dem Formular definiert sind, wieder via HTTP/CGI-Schnittstelle an den MapServer zurückgegeben. Dadurch werden bei jeder Anfrage des Nutzers für die Karte, die Übersichtskarte, die Legende und die Maßstabsleiste Bilddateien erzeugt, die an entsprechenden Stellen des Templates eingesetzt werden.

Zusätzlich kann die Funktion des Clients durch webfähige Sprachen wie JavaScript, PHP oder PERL erweitert werden. In unserem Beispiel ist ein JavaScript eingebunden:

```
<script language="JavaScript" type="text/javascript"
src="http://localhost/Schwalmtal/java/codesnips.js"></script>
```

Danach beginnt der eigentliche Body unserer HTML-Datei. Direkt wird das Formular angefangen: <!-- start Formular -->

```
<form method='post' action='/cgi-bin/mapserv.exe' name='mapserv'>
```

Die Methode „post“ besagt nur, dass die übergebenen CGI-Parameter in einer kleinen Datei übergeben werden. Testen Sie ruhig einmal den anderen Weg, indem Sie statt „post“ einfach „get“ eintragen und auf die Adressenleiste Ihres Browsers schauen.

Im weiteren werden einige Tabellen definiert. Interessant wird es erst wieder ab Zeile 87:

```
<a
href="javascript:document.butinfo.src='http://localhost/morocco/graphics/icon_info_1.gif'" ...
>
usw.
```

Hier stehen die Werkzeuge und sind jeweils mit einer JavaScript Funktion verknüpft. Suchen Sie diese in der Datei /morocco/java/codesnips.js

Die Richtungspfeile sind genauso eingebunden

```
<a href='javascript:move(-0.5,0.5)'>
...
<img border='0' src='http://localhost/../nw.gif' width='15' height='15'></a>
```

und mit der Funktion „move“ im Javascript verknüpft.

In Zeile 152 findet man nun den Verweis auf die Referenzkarte und einen ersten Hinweis darauf, wie unser MapServer denn nun eigentlich funktioniert:

```
<input name='ref' type='image' src='[ref]' >
```

In den eckigen Klammern steht [ref]. Schauen wir uns das Ganze doch einmal zur Laufzeit an: Tippen Sie in Ihren Browser http://localhost/morocco/av2ms_init.html ein und klicken Sie auf den Button. Eine Karte mit Referenzkarte, Legende und Maßstabsleiste erscheint. Klicken Sie nun mit der rechten Maustaste irgendwo auf die Seite und wählen Sie „Quelltext anzeigen“. Suchen Sie die entsprechende Stelle <input name='ref' type='image' src='[ref]'> im Quelltext (suchen Sie nach „Referenzkarte“). Dort steht nun so etwas:

```
<input name='ref' type='image' src="http://localhost/temp/moroccoref10613913431948.png" >
```

Der MapServer setzt also an die Stelle mit den eckigen Klammern dynamisch den Namen einer Bilddatei ein. Sie können feststellen, dass diese Datei tatsächlich existiert, indem Sie einfach die obige Bildquelle http://localhost/temp/moroccoref10613913431948.png

in die Adressleiste Ihres Browsers kopieren.

Suchen Sie nun in der HTML-Datei nach weiteren Einträgen src=[img] / [scale] / [scalebar] / [legend]. An diesen Stellen passiert genau dasselbe.

Das Abfragetemplate

In der MAP-Datei wird definiert, ob für einen Layer Abfragen zugelassen werden oder nicht. Dies geschieht, indem man für einen Layer das Keyword „TEMPLATE“ in der Layer-Definition angegeben wird:

```
LAYER
  NAME 'Bodenarten'
  DATA bodenarten
```

```
  TYPE Polygon
  TEMPLATE 'Bodenarten_query.html'
 ...
```

Die HTML-Datei 'Bodenarten_query.html' enthält alle weiteren Informationen, die der MapServer benötigt, um bei einer Abfrage die Ergebnisse des angeklickten Objektes zu präsentieren.

Im Header der Projektdatei kann darüber hinaus in der Querymap-Sektion ein Screenshot definiert werden, in dem eine Übersichtskarte des aktuellen Ausschnitts generiert und in das Template eingefügt wird. Es ist möglich, das abgefragte Objekt einzufärben.

```
QUERYMAP
 SIZE 200 200
 STATUS ON
 STYLE HILITE                    # Hier wird gesagt „Objekt einfärben"
  COLOR 255 0 0                  # Einfärbungsfarbe, hier: rot
END
```

Sonstige Templates

Um eine Anwendung möglichst benutzerfreundlich zu gestalten, hat der Entwickler die Möglichkeit, weitere Templates zu definieren. Ein zentrales ist z.B. das „Empty-Template". Im Header der MAP-Datei wird mit dem Keyword

```
EMPTY „Nothing.html"
```

ein Template definiert, welches dann angezeigt wird, wenn der Benutzer im Abfragemodus kein Objekt angeklickt hat. Existiert ein solches Empty-Template nicht, so erscheint lediglich diese MapServer-Fehlermeldung, mit der ungeübte Benutzer nicht viel anfangen können:

msQueryByPoint(): Search returned no results. No matching record(s) found.

*** * ***

Datenhaltung mit PostgreSQL/PostGIS

Von Christina Biakowski

Zusammenfassung

PostGIS bezeichnet die räumliche Erweiterung zur Speicherung und Verwaltung von Geodaten in der Open Source-Serverdatenbank PostgreSQL. Durch PostGIS kann PostgreSQL als räumliches Datenbank-Backend für GIS-Applikationen eingesetzt werden. Die Vorteile sind die u.a. die Mehrbenutzerfähigkeit und die Verwendung komplexer SQL-Abfragen zur Analyse von Geodaten. Die OGC Simple Features Spezifikationen sind in PostGIS über die GEOS Bibliothek vollständig implementiert. PostGIS/PostgreSQL bietet damit eine OGC-konforme Open Source-Alternative zu GIS Datenbank-Backends proprietärerer Anbieter. Vom UMN MapServer wird die Einbindung von PostGIS-Daten unterstützt.

Das Kapitel beschreibt zunächst das von PostGIS verwendete OGC-konforme Geodatenmodell und liefert einen Überblick über die implementierten GIS-Funktionen. Es folgen Hinweise zu Schnittstellen und Tools zum Datenaustausch sowie zum Einbinden von Geodaten, die in PostgreSQL/PostGIS vorgehalten werden, in UMN MapServer-Anwendungen.

Abschließend wird anhand eines einfachen Beispiels das Anlegen einer PostgreSQL Datenbank, das Laden der PostGIS-Erweiterung, der Import einer Shape-Datei in die neue Datenbank sowie das Einbinden der importierten Daten in eine MAP-Datei beschrieben.

1 Einführung

Die Datenverwaltung in GI-Systemen verschiebt sich zunehmend von proprietär strukturierten Dateisystemen hin zur Vorhaltung von Geodaten in relationalen Datenbanken. Die Vorteile liegen dabei auf der Hand: neben der gemeinsamen Verwaltung von Geo- und Sachdaten sowie der Möglichkeit, für die Analyse und Verwaltung der Geodaten das SQL-Interface der Datenbank nutzen zu können, ist v.a. die Mehrbenutzerfähigkeit solcher Systeme hervorzuheben. Redundante Datenhaltung und die damit stets verbundene Gefahr von Dateninkonsistenzen wird vermieden. Der Zugriff verschiedener Nutzer über das Intra- oder Internet kann über die Zugriffssteuerung der Datenbank kontrolliert werden. Des weiteren bieten gut organisierte relationale Datenbanken eine wesentlich höhere Leistung beim Zugriff auf Ausschnitte.

Als Open Source-Alternative zu GIS Datenbank-Backends proprietärer Anbieter ist die Verwaltung von Geodaten in der Serverdatenbank PostgreSQL mit der räumlichen Erweiterung PostGIS zu sehen.

Das objektrelationale Datenbankmanagementsystem PostgreSQL ist ursprünglich aus

einem universitären Projekt der University of California at Berkeley hervorgegangen. Inzwischen wird die Entwicklung von einer aus einem Firmenverbund zusammengesetzten Developer Group vorangetrieben. PostgreSQL zeichnet sich durch umfassende Funktionalitäten aus. Zu nennen sind u.a. die Unterstützung von referentieller Integrität, das Transaktionsmanagement sowie die Möglichkeit Stored Procedures und Trigger einzusetzen. Ein guter Artikel zu freien Datenbanken findet sich auch in der iX 2/2004.

Die Entwicklung von PostGIS - ebenfalls ein Open Source-Projekt – wird von Refractions Research Inc. aus Kanada geleitet.

Zusammen mit der PostGIS-Erweiterung stellt PostgreSQL ein leistungsstarkes Instrument zur Verwaltung von Geodaten dar. Die Einbindung von PostGIS-Daten wird vom UMN MapServer unterstützt.

2 Systemvoraussetzungen und Sourcen

PostgreSQL kann unter allen UNIX-kompatiblen Plattformen eingesetzt werden. Die Lauffähigkeit unter Windows ist für die nächste Version zugesichert. Bislang kann PostgreSQL unter Windows nur unter der UNIX-Emulation Cygwin eingesetzt werden.

Die aktuelle PostgreSQL-Version ist 7.4.1, das neueste PostGIS-Release ist Version 8.1 (Stand Januar 2004). Die Sourcen sowie ausführliche Installationsanleitungen sind über die jeweiligen Webseiten (s. Linkliste) zu beziehen (PostgreSQL: BSD-Lizensierung, PostGIS: GNU GPL-Lizensierung).

Für die Administration und Datenverwaltung in PostgreSQL stehen verschiedene Clients zur Verfügung. Die Installation von PostgreSQL umfasst automatisch den shell-basierten Client *psql*, der in Kapitel 7 kurz vorgestellt wird. Als Client mit grafischer Benutzeroberfläche steht für Windows- und UNIX-Systeme auf der PostgreSQL-Webseite *pgAdmin* zum Download zur Verfügung.

3 Geodatenmodell in PostGIS

3.1 OGC Simple Features Spezifikation

Die OGC "Simple Features Specification for SQL" definiert Standards für

- Geo-Objekttypen
- Funktionen zur Analyse und Manipulation von Geo-Objekten sowie
- Metadaten-Tabellen. Zur Gewährleistung der Konsistenz der Metadaten werden von der OGC Spezifikation ebenfalls spezielle Funktionen definiert wie z.B. das Hinzufügen und Löschen einer Spalte mit Geo-Objekten.

Seit Version 8 sind die OGC Simple Features Spezifikationen in PostGIS vollständig umgesetzt. Während in den Vorgängerversionen die definierten Geo-Objekttypen und

Metadaten-Tabellen bereits vollständig implementiert waren, fehlte zum großen Teil noch die Integration räumlicher Funktionen und Operatoren (wie z.B. Contains(), Buffer(), Difference() etc) I. Über die Integration der OGC-konformen GEOS-Bibliothek umfasst PostGIS jetzt auch diese OGC-Standards.

3.2 Geo-Objekttypen

Gemäß der OGC Simple Features Spezifikation können Geo-Objekte/Geometrie-Objekte in zwei Formaten ausgedrückt werden:

- im *Well-Known Binary* (WKB) – Format als Binary Large Object (BLOB)
- im "lesbaren" *Well-Known Text* (WKT) – ASCII Textformat als Array von Koordinatenwerten

Beide Formate enthalten neben den Koordinaten Informationen zum Geometrietyp. Intern speichert PostgreSQL/PostGIS die Geo-Objekte immer im (kleineren) WKB-Format. Als Geo-Objekttypen werden POINT, LINESTRING, POLYGON, MULTIPOINT, MULITILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION unterschieden. Geo-Objekte können 2- oder 3-dimensional sein.

Beispiele für im WKT-Format dargestellte Geo-Objekte:

```
POINT(2572292.2 5631150.7)

LINESTRING (2566006.4 5633207.9, 2566028.6 5633215.1, 2566062.3 5633227.1)

MULTILINESTRING((2566006.4 5633207.9, 2566028.6 5633215.1), (2566062.3 5633227.1,
2566083 5633234.8))

POLYGON (2568262.1 5635344.1, 2568298.5 5635387.6, 2568261.04 5635276.15, 2568262.1
5635344.1);

MULTIPOLYGON(((2568262.1 5635344.1, 2568298.5 5635387.6, 2568261.04 5635276.15,
2568262.1 5635344.1), (2568194.2 5635136.4, 2568199.6 5635264.2, 2568200.8 5635134.7,
2568194.2 5635136.4 )))
```

3.3 Feature Tables und Metadaten

Feature Tables

Die Geo-Daten werden in PostgreSQL/PostGIS in *Feature Tables* vorgehalten. Jede Feature Table enthält eine Geometrie-Spalte zur Speicherung der Geo-Objekte. Beim Anlegen der Tabelle muss der Geometrie-Typ (POINT, LINE etc.), der in der Feature Table gespeichert werden soll, angegeben werden. Die OGC-Spezifikation legt fest, dass

nur gleiche Geometrie-Typen (POINT, LINE etc.) gemeinsam in einer Feature Table verwaltet werden können. Soll die Feature Table unterschiedliche Geometrie-Typen enthalten, ist der allgemeine Typ GEOMETRY anzugeben.

Metadatentabellen *geometry_columns* und *spatial_ref_sys*

Zu jeder Feature Table gibt es einen Eintrag in der Metadatentabelle *geometry_columns*. Die Tabelle *geometry_columns* wird automatisch beim Laden der PostGIS-Funktionen in die PostgreSQL-Datenbank angelegt (das Laden der PostGIS-Erweiterung in eine PostgreSQL-Datenbank ist in Kapitel 7 „Einbinden von PostGIS-Daten in eine MapServer-Karte“ beschrieben). Die Metadatentabelle enthält Informationen zum Namen der Feature Table, dem Namen der Geometrie-Spalte, dem Geometrie-Typ, dem EPSG-Code des verwendeten Projektionssystems (SRID) u.a.

Der Eintrag im Feld SRID referenziert als Fremdschlüssel die zweite PostGIS Metadatentabelle *spatial_ref_sys*. Die Tabelle enthält im Feld *SRTEXT* die textliche Beschreibung der Projektionsparameter des jeweiligen EPSG-Codes. PostGIS unterstützt die Transformation von Koordinaten unter Nutzung der PROJ4-Bibliothek, auf deren Funktionen auch der UMN MapServer zugreift. Die von der PROJ4-Bibliothek benötigten Projektionsparameter sind im Feld *PROJ4TEXT* gespeichert.

Um die Konsistenz der Metadaten zu gewährleisten, sind gemäß OGC Spezifikation in PostGIS spezielle Funktionen zum Hinzufügen (*AddGeometryColumn()*) und Löschen (*DropGeometryColumn()*) einer Geometrie-Spalte einer Feature Table implementiert. Über diese Funktionen wird automatisch ein Eintrag in der Metadatentabelle *geometry_columns* eingefügt bzw. wieder entfernt.

Anlegen und Löschen einer Feature Table

Das Anlegen einer Feature Table besteht aus 2 Schritten:

1. Anlegen einer Tabelle (CREATE TABLE-SQL Statement)
2. Hinzufügen der Geometrie-Spalte mit der Funktion AddGeometryColumn():

   Syntax:

```
AddGeometryColumn(<db_name>, <table_name>, <column_name>,
<srid>, <type>, <dimension>)
```

mit

- <db_name> – Name der Datenbank
- <table_name> – Name der Feature Table
- <column_name> – Name der Geometrie-Spalte, die hinzugefügt werden muss
- <srid> – EPSG-Code des verwendeten Projektionssystems
- <type> - Geometrie-Typ (POINT; LINESTRING, POLYGON, MULTIPOINT,

MULITILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION, GEOMETRY – anzugeben, wenn eine Tabelle unterschiedliche Geometrie-Typen enthalten soll)

- <dimension> - Dimension der Daten (2 oder 3)

Über diese Funktion wird automatisch ein Eintrag zu der Feature Table in der Metadatentabelle *geometry_columns* vorgenommen.

Analog dazu besteht das Löschen einer Feature Table ebenfalls aus 2 Schritten:

1. Löschen der Geometrie-Spalte mit der Funktion DropGeometryColumn()

Syntax:

```
DropGeometryColumn(<db_name>, <table_name>, <column_name>)
```

Damit wird die Geometrie-Spalte aus der Feature Table gelöscht und der zugehörige Eintrag in der Metadatentabelle *geometry_columns* entfernt.

2. Löschen der Feature Table (DROP TABLE-SQL Statement)

Geometriedaten in Feature Table einfügen

Die Feature Tables können durch übliche SQL-INSERT-Statements mit Daten befüllt werden. Zum Einfügen von Geo-Objekten ist die Funktion *GeometryFromText()* zu verwenden.

Syntax:

```
GeometryFromText(<geometry>, <SRID>)
```

<geometry> bezeichnet die Angaben zum jeweiligen Geo-Objekt im WKT-Format.

Beispiel:

In der Datenbank *geodaten1* soll eine Feature Table *baum* angelegt werden, in der Bäume als punkthafte Objekte gespeichert werden. Als Projektionssystem wird Gauss Krüger (3. Streifen) verwendet. Folgende SQL-Strings werden eingegeben:

```
create table baum (gid int4, baum_typ varchar);

select AddGeometryColumn('geodaten1', 'baum',
'the_geom', '31493','POINT',2);

insert into baum values ('1','Erle',GeometryFromText
('POINT(3564780.70 5631558.75)', 31493));

insert into baum values ('2', 'Linde',
GeometryFromText('POINT(3564850.72 5631672.23)',
31493));
```

4 Räumliche PostGIS-Funktionen

In PostGIS sind die in den OGC Simple Features Spezifikationen definierten Funktionen zur Abfrage und Manipulation von Geodaten wie z.B. *intersects()*, *touches()*, *crosses()*, *overlaps()*... vollständig implementiert. Damit wird ein umfassendes Instrumentarium zur Durchführung komplexer Geodaten-Analysen und -Manipulationen bereitgestellt.

In folgendem einfachen Beispiel wird die Bounding Box der Feature Table *nutzung* abgefragt:

```
geodata2=# SELECT EXTENT(the_geom) FROM nutzung;
                    extent
-------------------------------------------
 BOX3D(4426901 5466454 0,4448042 5489927 0)
(1 row)
```

Eine Beschreibung aller verfügbaren räumlichen Funktionen ist der PostGIS Online-Dokumentation zu entnehmen (s. Linkliste).

5 Austausch von Geodaten mit PostGIS

Zum Import vorhandener Geodaten stehen verschiedene Tools zur Verfügung. PostGIS umfasst ein Programm für den Im- und Export von ESRI Shapedateien (der *shp2pgsql* Shapeloader wird in Kapitel 7 noch genauer vorgestellt).

Die *ogr2ogr*-Funktion der *OGR-Utilities* (siehe Linkliste) ermöglicht die Konvertierung von Geodaten von und nach PostgreSQL/PostGIS, die in den Formaten TIGER, ESRI

Shapefiles, MapInfo sowie GML vorgehalten werden. Die Open Source OGR-Bibliothek umfasst Funktionen für den Zugriff auf zahlreiche Vektorformate und wird auch vom UMN MapServer verwendet. Die OGR-Bibliothek ist Teil der GDAL-Bibliothek.

Zum Import von SICAD/SD c60-Dateien kann das OpenSource-Projekt *C60-PostGIS-Migrator* der Firma CCGIS verwendet werden.

Von externen Anwendungen kann über die Schnittstellen ODBC und JDBC auf PostgreSQL/PostGIS zugegriffen werden. Daneben ist mit verschiedenen Programmiersprachen (z.B. PHP) ein nativer Zugriff auf die Datenbanken möglich.

Ein interessante Alternative zum Kennenlernen ist die Möglichkeit, PostgreSQL Tabellen über ODBC Datenquellen in MS Access einzubinden.

6 PostGIS und UMN MapServer

Der UMN MapServer kann in PostgreSQL/PostGIS vorgehaltene Geodaten als Datenformat unterstützen. Ob die installierte MapServer-Version PostGIS unterstützt, kann über die Eingabe von *mapserv -v* auf der Kommandozeile überprüft werden. Die zurückgegebenen Versions-Informationen müssen den Eintrag "INPUT=POSTGIS" enthalten. Ist dies nicht der Fall, muss MapServer mit der entsprechenden PostGIS-Unterstützung neu kompiliert bzw. andere Binaries verwendet werden.

Für den Fall, dass trotz PostGIS-Unterstützung mit MapServer keine PostGIS-Daten dargestellt werden können, steht auf der MapServer-Webseite unter http://mapserver.gis.umn.edu/cgi-bin/wiki.pl?PostGIS eine Liste mit Tipps für die Fehlersuche zur Verfügung. Des weiteren besteht die Möglichkeit, Fragen auf der UMN MapServer Userliste zu posten.

Für die Einbindung von PostgreSQL/PostGIS-Daten als Layer in einer MAP-Datei sind PostGIS-spezifische Angaben zu den Parametern CONNECTIONTYPE, CONNECTION und DATA erforderlich:

```
LAYER
   ...
   CONNECTIONTYPE postgis
   CONNECTION "user=[username] dbname=[database] host=
   [server] port=5432 password=[password]"
   DATA "[geometry_column] from [table]"
   ...
END
```

Über den FILTER-Parameter kann das Abfrageergebnis eingegrenzt werden:

```
FILTER "[filteritem] = [Kriterium]"
```

Komplexe Abfragen über mehrere Tabellen können über die Verwendung von Subselects im DATA-String generiert werden, wie in folgendem Beispiel, in dem die Geometrien der Straßen (Tabelle *str_geo*) in einer anderen Tabelle vorliegen als die Straßennamen (Tabelle *str_nam*). Über ein JOIN werden die Geometrien mit Daten aus der Tabelle *str_nam* beschriftet:

```
DATA "the_geom from (select str_geo.gid as gid,
str_geo.the_geom as the_geom, str_nam.street_name from
str_geo LEFT JOIN str_nam ON str_geo.kzf = str_nam.kzf)
as foo using SRID=31494 using unique gid"
```

UMN MapServer benötigt die Angabe des Projektionssystems (SRID) sowie einer eindeutigen ID (*using unique*).

7 Beispiel - Einbinden von PostGIS-Daten in eine MapServer-Karte

In diesem Kapitel wird anhand eines einfachen Beispiels Schritt-für-Schritt das Anlegen einer PostgreSQL Datenbank, das Laden der PostGIS-Erweiterung, der Import einer Shape-Datei in die neue Datenbank sowie schließlich das Anzeigen der angelegten Daten in einer UMN MapServer-Karte beschrieben.

7.1 Anlegen einer PostGIS-Datenbank

1. Anlegen der PostgreSQL-Beispieldatenbank *beispieldb*

Um eine Datenbank zu erzeugen, ist auf der Kommandozeile folgender Befehl einzugeben:

```
createdb beispieldb
```

Die im Zusammenhang mit dem createdb-Befehl verfügbaren Optionen können dem "PostgreSQL Reference Manual" entnommen werden, das auf der PostgreSQL-Seite zum Download zur Verfügung steht.

2. Laden der Sprache PL/pgSQL

PostgreSQL ermöglicht die Erstellung eigener Funktionen mit verschiedenen prozeduralen Sprachen, die allerdings nicht per Default zur Verfügung stehen, sondern ausdrücklich geladen werden müssen. Für die PostGIS-Erweiterung wird die Sprache

PL/pgSQL benötigt. Diese wird in die Datenbank geladen über:

```
createlang plpgsql beispieldb
```

3. Laden der Datei *postgis.sql*

   Die PostGIS-Installation umfasst die Datei *postgis.sql*, welche die PostGIS Objekt- und Funktions-Definitionen enthält. Über „Laden der Datei“ werden die entsprechenden Objekte und Funktionen in der Datenbank verfügbar. Gleichzeitig wird die Metadaten-Tabelle *geometry_columns* angelegt. Das Laden in die Beispiel-Datenbank *beispieldb* erfolgt über:

   ```
   psql -d beispieldb -f postgis.sql
   ```

(Zum Befehl *psql* siehe auch Kapitel 7.3).

4. Laden der Datei *spatial_ref_sys.sql*

   Zum Erzeugen der Metadaten-Tabelle *spatial_ref_sys*, welche die EPSG-Codes mit den dazugehörigen Projektionsparametern enthält, muss die Datei *spatial_ref_sys.sql* geladen werden:

   ```
   psql -d beispieldb -f spatial_ref_sys.sql
   ```

7.2 Import einer Shapedatei mit dem PostGIS-Shapeloader

In der PostGIS-Installation enthalten ist ein Shapeloader (*shp2pgsql*) und ein Shapedumper (*pgsql2shp*). Die Programme ermöglichen es, ESRI Shapefiles nach PostGIS zu importieren bzw. umgekehrt PostGIS-Daten in Shapefiles zu konvertieren.

Der *shp2pgsql*-Shapeloader konvertiert die Shapefiles für das Einfügen der Daten in die PostgreSQL/PostGIS-Datenbank in die dafür erforderlichen SQL-Statements (CREATE TABLE..., AddGeometryColumn() etc. sowie INSERT-Statements). Die dabei erzeugte Input-Datei kann anschließend in die Datenbank geladen werden. Die Geo-Objekte werden im WKT-Format in die Datenbank geschrieben und intern als WKB gespeichert.

Syntax zum Aufruf des *shp2pgsql*-Shapeloaders:

```
shp2pgsql [<options>] <shapefile name> <table name>
<database>
```

mit folgenden Optionen:

(-d | -a | -c – Optionen schließen sich gegenseitig aus)

-d – Tabelle wird gelöscht, neu erzeugt und mit den Daten des angegebenen Shapefiles befüllt.

-a – Shapefile wird an Tabelle angehängt. Dabei muss die Tabellenstruktur übereinstimmen.

-c – Default. Erzeugt neue Tabelle und befüllt diese mit den Daten des Shapefiles.

-D - Erzeugt neue Tabelle und befüllt diese mit den Daten des Shapefiles. Dabei wird statt dem default "INSERT" SQL Format das PostgreSQL "dump"-Format (copy) verwendet. Diese Option ist bei sehr großen Shapedateien interessant, da das Einfügen der Daten in die Datenbank darüber wesentlich schneller erfolgt.

-s <srid> - EPSG-Code. Bei fehlender Angabe wird -1 verwendet.

Beispiel:

In die Datenbank *beispieldb* soll das Shapefile *baum.shp* (Punktdaten, Projektionssystem Gauss-Krüger 3. Streifen) importiert werden. Die Daten sollen in eine Feature Table namens *baum* geschrieben werden. Die vom Shapeloader bei der Konvertierung erzeugten SQL-Statements werden in die Input-Datei *baum.sql* geschrieben:

```
shp2pgsql -s 31493 baum baum beispieldb > baum.sql
```

Anschließend wird die Input-Datei in die Datenbank geladen:

```
psql -d beispieldb -f baum.sql
```

Unter Verwendung der UNIX-Pipes können Konvertierung und Laden in die Datenbank auch in einem Schritt durchgeführt werden:

```
shp2pgsql -s 31493 baum baum beispieldb | psql -d beispieldb
```

In der Datenbank wurde nun die Feature Table *baum* angelegt sowie ein Eintrag in der Metadaten-Tabelle *geometry_columns* vorgenommen.

7.3 Ansicht der eingefügten Daten in PostGIS

In Kapitel 2 wurde bereits auf die unterschiedlichen Clients hingewiesen, die für die Arbeit mit PostgreSQL-Datenbanken zur Verfügung stehen. Die PostgreSQL-Installation umfasst automatisch den Terminal-basierten Client *psql*, der hier kurz vorgestellt werden soll.

Eine umfassende Beschreibung der Client-Funktionalitäten ist dem "PostgreSQL Reference Manual" (Teil "PostgreSQL Client Applications") zu entnehmen, das auf der PostgreSQL-Seite zum Download zur Verfügung steht.

Über

```
psql beispieldb
```

wird eine Verbindung zu unserer Beispiel-Datenbank hergestellt und das interaktive PostgreSQL Datenbank-Terminal geöffnet. Hier können nun zur Abfrage und Manipulation der Daten SQL-Statements eingegeben werden. Des weiteren stehen in *psql* eine Reihe an Metakommandos zur Verfügung, die jeweils mit einem Backslash beginnen. Hier ein Auszug einiger wichtiger Metakommandos:

```
\h syntaktische Hilfe zu SQL-Befehlen
\? Anzeige aller Kommandos
\q Beenden von psql
\g oder ; Abschließen einer SQL-Anweisung
\r Leeren des Abfragepuffers
\dt Anzeige aller Tabellen
\d [name] Tabelle oder Index mit [name] beschreiben
\x Wechsel zwischen Anzeigemodus (lang/kurz)
\l Anzeige aller Datenbanken
```

Über die Eingabe von

```
\d baum
```

wird die Tabellenstruktur der Feature Table angezeigt, die mit Hilfe des Shapeloaders erzeugt wurde. Beim Import eines Shapefiles mit dem Shapeloader wird die Geometrie-Spalte automatisch *the_geom* benannt. Zusätzlich wird das Feld *gid* angelegt und mit einer eindeutigen ID befüllt.

7.4 Räumlicher Index (GiST)

Sobald Feature Tables die Größe von einigen tausend Datensätzen erreichen, ist die Vergabe eines räumlichen Index im Hinblick auf eine gute Performance unerlässlich.

In PostgreSQL/PostGIS ist der GiST (General Search Tree) – Index als räumlicher Index implementiert.

Syntax zur Erstellung eines GiST-Index:

```
CREATE INDEX [indexname]
ON [tablename]
USING GIST ( [geometrycolumn] GIST_GEOMETRY_OPS );
```

Dabei verweist "gist_geometry_ops" auf eine spezielle Gruppe von Vergleichsoperatoren ("gist_geometry_ops" ist Teil der PostGIS-Erweiterung), die der Datenbankserver bei der Index-Erstellung einsetzt.

In folgendem Beispiel wird für die Feature Table *baum* ein GiST-Index namens *geoidx* erstellt:

```
CREATE INDEX geoidx
ON baum
USING GIST ( the_geom GIST_GEOMETRY_OPS );
```

Anschließend ist für die Feature Table der Befehl

```
VACUUM ANALYSE baum
```

aufzurufen, damit der PostgreSQL-Datenbankserver seine Spaltenstatistiken aktualisiert und der neu gesetzte Index bei der Abfrageoptimierung berücksichtigt wird.

7.5 Einbindung in die MAP-Datei

Die Einbindung der erzeugten Feature Table *baum* in eine MAP-Datei könnte folgendermaßen aussehen:

```
LAYER
   NAME "Baumdaten"
   TYPE POINT
   ...
    CONNECTIONTYPE postgis
    CONNECTION "user=myuser dbname=beispieldb
   host=144.222.9.99 port=5432"
    DATA "the_geom from baum"
   ...
   CLASS
   ...
   END
END
```

8 Link-Liste

PostgreSQL

PostgreSQL Webseite:

http://www.postgresql.org/

PostgreSQL und Windows:

http://techdocs.postgresql.org/guides/Windows

Online-Version des Buches "PostgreSQL: Introduction and Concepts" von Bruce Momjian

http://www.postgresql.org/docs/aw_pgsql_book/index.html

Offizielle PostgreSQL-Dokumentation als deutsche Übersetzung:

http://www.postgresql.org/docs/pg_handbuch.html

Deutschsprachige Seite zu PostgreSQL:

http://www.postgres.de/

Deutschsprachige Einführung in PostgreSQL – Buchtipp:

Boenigk, Cornelia: PostgreSQL. Grundlagen, Praxis, Anwendungsentwicklung mit PHP. Heidelberg, 2003. 412 S.

PostGIS

PostGIS Webseite:

http://postgis.refractions.net/

PostGIS Online Dokumentation:

http://postgis.refractions.net/docs/

PostGIS und UMN MapServer:

http://postgis.refractions.net/docs/x556.html

OGR Utility Programs (Konvertierung von Geodaten):

http://gdal.velocet.ca/projects/opengis/ogrhtml/ogr_utilities.html

UMN MapServer

Check-Liste zum Debuggen von MapServer-PostgreSQL-PostGIS Connections:

http://mapserver.gis.umn.edu/cgi-bin/wiki.pl?PostGIS

* * *

AveiN! Die UMN MapServer Erweiterung für ArcView GIS©

Von Till Adams

Da in Kürze eine aktualisierte Version (AveiN! 1.4) mit dem entsprechenden Handbuch herausgegeben wird, finden Sie hier lediglich einige einführende Worte zum Funktionsumfang von AveiN!

Mit AveiN! (ArcView© einfach ins Netz!) können Sie GIS-Karten aus ArcView GIS 3.x©) einfach und schnell im Intra- oder Internet veröffentlichen.

MapServer Erweiterung für ArcView GIS

Was ist AveiN! ?

Diese ArcView GIS©-Erweiterung fungiert als Schnittstelle zwischen dem lokalen ArcView GIS© und dem UMN MapServer. Sie erstellt die erforderlichen Dateien, um GIS-Karten in Verbindung mit dem *UMN MapServer* sowie dem ebenfalls frei verfügbaren *Apache WebServer* zu einer WebGIS-Anwendung zu machen.

Der UMN MapServer ist eine Open Source-basierte Software, die weltweit von einer großen Benutzergruppe weiter entwickelt und gepflegt wird. Er arbeitet als CGI-Script plattformunabhängig in verschiedenen WebServern und bietet damit eine einfache und vor allem kostengünstige Lösung, um dynamische Karten im Internet bereit zu stellen.

AveiN! Funktionalitäten

AveiN! erstellt aus GIS- Karten die erforderlichen Dateien, um diese mit Hilfe des *UMN MapServers* im Intra-/Internet zu veröffentlichen. Dazu muss die Systemumgebung mit dem UMN MapServer und einem WebServer, z.B. der freien Software Apache Webserver, eingerichtet sein.

AveiN! übernimmt alle Vektorthemen sowie Rasterbilder als Tiff-Dateien aus dem View. Flächenthemen und Linienfarben werden direkt übernommen. Aufgrund der grund-verschiedenen Symbolisierung in ArcView und dem UMN MapServer bietet AveiN! eine große Anzahl an Punkt- und Liniensymbolen zur Auswahl an. Weitere Symbole können in AveiN! integriert werden.

AveiN! schreibt Ihnen die Mapdatei so heraus, daß Sie diese direkt mit dem GetCapabilities Request in die CCGIS Client-Suite Mapbender (Freie Software) einbinden können, wodurch unter anderem eine Benutzerverwaltung ermöglicht wird.

Weiter Informationen: http://www.terrestris.de/links/downloads.html

* * *

Kompilieren und Betrieb von UMN MapServer auf verschiedenen Plattformen

Von Benjamin Thelen

Vorneweg

Dieses Kapitel hegt nicht den Anspruch auf Vollständigkeit. Es dient als Leitfaden zum weiteren Lesen, Ausprobieren und um einen ersten Einstieg in diese umfangreiche Thematik zu erhalten.

Weitere Hilfe bieten zahlreiche Bücher, im Internet verfügbare Dokumentationen, Wiki-Pages, How-To's, 'manual pages' sowie Mailinglisten, derer es zu jeder Software zum Teil mehrere nach Unterthemen geordnet gibt.

Begriffe wurden soweit sinnvoll und möglich in die deutsche Sprache übertragen.

Das Thema Support möchte ich an dieser Stelle anschneiden, weil es ein häufig angeführtes Argument zur Entscheidung für die Distribution eines kommerziellen Anbieters ist und projektorientierte Derivate von vornherein gar nicht erst in Betracht gezogen werden.
Je nach Zielsetzung und Besonderheiten der eigenen Umgebung befindet man sich ziemlich schnell außerhalb der Gewährleistung. Der Installationssupport der SuSE Linux AG beschränkt sich nur auf das Notwendigste. Dafür ist er aber auch kostenlos. Einem geschenkten Gaul... Alles was darüber hinaus geht, wird über eine kostenpflichtige 0190er Nummer abgewickelt. Das betrifft z.B. bereits das Anlegen weiterer Partitionen, die Auswahl anderer Pakete, als die der Standardauswahl oder gar die manuelle Paketinstallation mit dem Programm *rpm*. Das ist sicherlich aus Sicht des Anbieters nachvollziehbar, jedoch ganz bestimmt kein Argument, um sich für die Distribution eines kommerziellen Anbieters entscheiden zu müssen. Red Hat hat seine Produkte für den Privatanwender inzwischen vollständig eingestampft, der Support läuft aus. Auch der Enterprise Support bewegt sich natürlich im Rahmen zuvor vereinbarter Gewährleistung. Unser Ziel, einen Web Map Server aufzusetzen, sprengt diesen ziemlich schnell. Vielleicht ändert sich das mit dem Engagement von IBM und Co. Allerdings funktioniert Freie Software ein bisschen anders und gerade das macht die Stärke der *Community* wohl auch aus.

Wer sich aber hiermit auseinandersetzt, wird schnell den 24 Stunden Support von Mailinglisten, Wiki-Pages und umfangreicher Dokumentationen im Netz zu schätzen lernen. Der Wissenspool zahlreicher Anwender bietet im Gegensatz zum kommerziellen Support zwangsläufig weit bessere Unterstützung. Allerdings ist auch Eigeninitiative gefragt.

Diese Überlegungen sollten auch bei der Entscheidung für ein Betriebssystem berücksichtigt werden. Die besten Erfahrungen haben wir mit dem freien UNIX®-

ähnlichen FreeBSD und dem freien debian GNU/Linux gemacht. Beide bieten keinen kommerziellen Support durch ein "Mutterhaus", sind aber wesentlich besser grundausgestattet als ihre kommerziellen Pendants. Das liegt vor allem auch daran, dass SuSE & Co. großen Wert auf Zertifizierungen legen – ein Prozess, der den Eingang neuer Entwicklungen in das Betriebssystem stark verzögern kann.

Wer dennoch Geld ausgeben möchte und einen Ansprechpartner, den er treten kann, findet zahlreiche Systemhäuser, die auch für *BSD (FreeBSD, OpenBSD, NetBSD) und debian GNU/Linux Unterstützung anbieten.

Die Zielsetzung

Dieses Kapitel beschreibt am Beispiel des freien UNIX®-ähnlichen Betriebssystems FreeBSD alle erforderlichen Installationsschritte, um eine WMS-fähige UMN MapServer Implementierung mit PostgreSQL/PostGIS Datenbankanbindung und der OGC-kompatiblen browserbasierten Client-Suite Mapbender bereitzustellen. Diese Komponenten bilden das Rückgrat einer sehr leistungsfähigen GDI und können für viele WebGIS Anwendungsfälle eingesetzt werden.

FreeBSD ist ein UNIX®-ähnliches Betriebssystem, welches von 4.4BSD abstammt, der UNIX® Version, die an der Universität von Kalifornien, Berkeley entwickelt wurde und eine optimale Plattform für die Bereitsstellung eines Servers bietet. Zahlreiche Anwender, unter ihnen so prominente wie Nokia, Yahoo! und zahlreiche ISPs, wie z.B. UUNet und Nacamar, setzen FreeBSD ein.

Zusätzlich werden Hinweise zu SuSE Linux 9.0 und debian GNU/Linux 3.woody gegeben

Selbstverständlich ist auch der Einsatz anderer Derivate möglich. Hier seien beispielsweise die freien UNIX®-ähnlichen Systeme OpenBSD und NetBSD genannt, aber auch nicht-offene kommerzielle Systeme, wie SUN Solaris, das auf dem BSD-UNIX® basierende Mac OS X oder die kommerziellen Linux Derivate Red Hat, Mandrake Linux, United Linux/SuSE Linux Enterprise Server, und, nicht zu vergessen, das auf Red Hat basierende freie Fedora Project.

1. Software Übersicht

http://mapserver.gis.umn.edu/doc40/unix-install-howto.html#obtain

1.1. Erforderliche Software

Die in diesem Buch beschriebene Installation wird ausschließlich mit freier bzw. quelloffener Software vorgenommen, die im Internet zum Herunterladen zur Verfügung

steht.

Die Zusammenstellung der Softwarepakete ermöglicht eine lauffähige Umgebung. Die Verwendung neuerer Softwareversionen ist eher möglich, als die älterer.

FreeBSD 4.9 http://www.freebsd.org

--> Betriebssystem. Unix®-ähnliches Betriebssystem.

debian GNU/Linux http://www.debian.org

--> Betriebssystem auf Basis des Linux-Kernels.

postGIS-0.7.5 http://postgis.refractions.net

--> Geo-spatial Aufsatz für PostgreSQL.

UMN-MapServer-3.6.6/4.0.1 http://mapserver.gis.umn.edu

--> Web Map Server der University of Minesota.

Apache 1.3.29 http://httpd.apache.org

--> Ein http Webserver.

php-4.3.4 http://www.php.net

--> Für die php-MapScript Unterstützung unbedingt als cgi bzw. für die Mapbender Client Suite erforderlich.

PostgreSQL-7.3.5 http://www.postgresql.org

--> Relationale Datenbank für den Einsatz von PostGIS.

MySQL-server 4.0.15 http://www.mysql.org

--> Relationale Datenbank für den Einsatz der Mapbender Client Suite.

gd-2.0.12 http://www.boutell.com/gd/

--> PNG und /oder GIF Rendering.

proj-4.4.5 http://www.remotesensing.org/proj/

--> Zur Unterstützung der On-the-fly Umprojizierung erforderlich. Für die WMS Unterstützung, mindestens Version 4.4.3.

libwww-5.4.0 http://www.w3.org/Library/

--> Zur Unterstützung von WMS Client Verbindungen. Optional, jedoch empfohlen für die WMS-Unterstützung. Nur UMN-MapServer 3.

curl-7.10.7 http://curl.haxx.se/libcurl/

--> Zur Unterstützung von WMS/WFS Client Verbindungen. Erforderlich bei WMS/WFS Unterstützung. Nur UMN-MapServer 4. Ersetzt libwww.

libiconv-1.9.1 http://www.gnu.org/software/libiconv/

-> Nur UMN-MapServer 4.

png-1.2.5 http://www.libpng.org/pub/png/

--> Unterstützung des PNG Ausgabeformats.

freetype2.1.3 http://www.freetype.org/

--> Unterstützung von TrueType Schriften.

gdal/ogr-1.1.8 http://www.remotesensing.org/gdal/

--> ogr: Zur Unterstützung der Ein- und Ausgabe zahlreicher Vektor Formate.

--> gdal: Zur Unterstützung der Ein- und Ausgabe zahlreicher Raster Formate.

jpeg-6b http://www.ijg.org/

--> Unterstützung des JPEG Ausgabeformats.

tiff-3.5.7 http://www.libtiff.org/

--> Unterstützung des TIFF Eingabeformats.

GEOS-1.0 http://geos.refractions.net/

--> - Geometry Engine - Open Source. PostGIS Extension.

pdflib 4.0.3 http://www.PDFlib.com

--> Zur Unterstützung des PDF Ausgabeformats. Achtung: Es gibt Lizenzbeschränkungen, da pdflib nicht vollständig Open Source ist. Alternativ ist die PDFlib Lite frei erhältlich.

Xerces-c-1.6.0/2.3.0 http://xml.apache.org/xerces-c/index.html

-->Begrenzte Unterstützung für das Lesen und Schreiben von GML-Dateien.

LibGeoTIFF http://www.remotesensing.org/geotiff/geotiff.html

--> Zur Unterstützung von GeoTIFF (Georeferenzierte TIFF Bilder).

Oracle Spatial Client Libraries http://www.oracle.com

--> Diese Bibliotheken werden mit der Oracle Datenbank ausgeliefert und sind erforderlich, um mit UMN MapServer Zugriff auf Oracle Daten zu erhalten.

SDE Client Libraries http://www.esri.com

--> Diese Bibliotheken werden mit ESRIs Spatial Data Warehouse ArcSDE ausgeliefert.

Mapbender WMS Client Suite www.mapbender.org

--> PHP4 basierte WMS Client Suite.

2. Software Installation

2.1. Allgemeines

Die Softwareinstallation auf den unterschiedlichen Unix®/Linux Derivaten unterscheidet sich deutlich durch die den jeweiligen Derivaten zur Verfügung stehenden

Softwarepakete, sowie durch die verwendeten Paketverwaltungssysteme. Auf allen Systemen besteht neben den verwendeten Paketverwaltungssystemen die Möglichkeit, Quellcode zu kompilieren und den kompilierten Quellcode zu installieren. Das bedeutet jedoch, dass Abhängigkeiten, also Softwarevoraussetzungen die auf dem Betriebssystem gegeben sein müssen, nicht automatisch erkannt - „aufgelöst“ - und installiert werden, sondern zuvor installiert worden sein müssen.

Sofern es die Möglichkeit gibt, eine vordefinierte Installationsauswahl vor der Betriebssysteminstallation zu treffen, sollte etwas in der Art von „Developer“ oder „Entwickler“ ausgewählt werden. Eine graphische Benutzeroberfläche ist zum einen nicht erforderlich und hat zum anderen nichts auf einem Serversystem zu suchen.

Auch unter Windows ist das Kompilieren von quelloffener Software möglich und bei speziellen Anforderungen, den UMN-MapServer betreffend, auch notwendig. Allerdings ist hier einiges an Wissen über den verwendeten Kompilierer, sowie über die zu kompilierenden Softwarepakete und deren Zusammenspiel erforderlich. Es wird in diesem Buch darauf nicht eingegangen, da das zum einen den Rahmen bei weitem sprengen würde und zum anderen auf vorkompilierte dll's und Binaries zurückgegriffen werden kann, die den meisten Anforderungen gerecht werden.

2.2. Softwareverwaltungssysteme

Um einen leichteren Einstieg in die Installation von Software zu verschaffen, wird ein kurzer Einblick gegeben. Für einen tieferen Einblick wird auf weiterführende Literatur verwiesen.

Grundsätzlich ist die Verwendung eines solchen Systems sehr zu empfehlen, da auf das Zielsystem angepasste Softwarepakete bzw. Patches installiert werden und so Systemstabilität gewährleistet werden kann, Abhängigkeiten mitinstalliert werden, eine leichte und vollständige Deinstallation möglich ist und z.B. auch Pfade zu Bibliotheken automatisch gesetzt werden.

SuSE Linux, Red Hat, Mandrake

Red Hat Linux, Mandrake Linux und SuSE Linux bieten innerhalb der standardmäßig installierten Fenstermanager (Gnome oder KDE) ein graphisches Werkzeug zur Softwareverwaltung an. Bei SuSE Linux ist das der *Yast2.* Diese graphischen Werkzeuge dienen als Frontend für das von diesen Linux-Systemen verwendete RPM-Paketverwaltungssystem (Red Hat Package Manager). Sie listen die installierten bzw. installierbaren Pakete auf, lösen Abhängigkeiten auf und lassen es zu, per Mausklick Softwarepakete zu installieren oder zu deinstallieren.

Auch auf der Kommandozeile bietet SuSE Linux eine graphische Benutzeroberfläche an, mit der auf ähnliche Art und Weise, allerdings ohne Maus, eine Softwareverwaltung möglich ist.

Das den graphischen Werkzeugen (Frontend) zugrundeliegende RPM-System lässt sich selbstverständlich auch auf der Kommandozeile mit Hilfe des Programmes *rpm* bedienen. Die *Manual Page* gibt Auskunft über alle Optionen. Die wichtigsten seien hier genannt.

rpm -i *paketname.rpm* installiert ein Paket von lokaler Quelle (z.B. von CD-ROM)
z.B: rpm -i /mnt/dvd/suse/src/postgresql-7.2.2-16.src.rpm

rpm -i *http://URLzuPaketname.rpm* installiert ein Paket direkt aus dem Internet
z.B: rpm -i ftp://ftp.gwdg.de/pub/linux/suse/ftp.suse.com/suse/i386/8.1/suse/i586/apache-1.3.26-53.i586.rpm

rpm -e *paketname* entfernt ein installiertes Paket
z.B: rpm -e apache (ohne Versionsnummer!)

rpm -qa [| grep *Suchwort*] listet alle installierten Pakete auf
z.B: rpm -qa |grep post liefert alle Pakete zurück, die mit „post“ anfangen.

Jedoch gibt es wesentlich mächtigere Alternativen, wie z.B. kommandozeilenbasierte rpm-Frontends, die im Gegensatz zum Programm *rpm* die Installation von Abhängigkeiten automatisch übernehmen. Hier seien apt/rpm (http://linuxwiki.de/apt/RPM bzw. http://linuxwiki.de/apt_2fSuSE) und yum (http://linuxwiki.de/yum) zu nennen. Beide Werkzeuge lehnen sich stark an die sehr leistungsfähigen *apt-tools* von debian GNU/Linux an. Ein weiteres Frontend ist das von Mandrake Linux bekannte urpmi (http://www.linuxwiki.de/urpmi).

Um den genauen Paketnamen eines Pakets herauszufinden, besteht u.a. die Möglichkeit, entweder die Installations-CD/DVD, die Homepage der SuSE Linux AG (www.suse.com), oder aber weitere Suchoptionen im graphischen Frontend oder auf der Kommandozeile heranzuziehen.

debian GNU/Linux

http://www.openoffice.de/linux/buch/

Um einem Vorurteil gleich zu Beginn den Wind aus den Segeln zu nehmen: Die Installation eines debian GNU/Linux ist erstens schnell, weil es ein schlankes System ist, und zweitens nicht schwerer zu durchzuführen als andere Systeminstallationen. Es ist anders. Um das gleiche Ziel zu erreichen, ein Serversystem für den Einsatz von UMN-MapServer bereitzustellen, unterscheidet es sich im Schwierigkeitsgrad nicht von anderen Systemen.

debian GNU/Linux bringt ein kommandozeilen-basiertes Frontend zu *dpkg*, dem nativen debian GNU/Linux Softwareverwaltungssystem, mit dem Namen *apt-get* (http://www.linuxwiki.de/apt) mit. *dpkg* ist das Pendant zu *rpm* und *apt-get* das Pendant zu *urpmi,yum* und Co bzw. Vorbild für diese.

apt-get update — aktualisiert die Paketverwaltungsdatenbank
Sollte ein Mal vor jeder Softwareinstallation ausgeführt werden, max.einmal am Tag.

apt-get upgrade — aktualisiert die Software und das System nach einem 'apt-get update'.

apt-get install *paketname* — installiert ein Paket
z.B: apt-get install apache

apt-get remove *paketname* — entfernt ein Paket
z.B: apt-get remove apache

apt-cache search *suchtext* — durchsucht die Paketverwaltungsdatenbank
z.B: apt-cache search postgres

Ob ein Paket von einer entfernten Quelle, z.B. dem Internet, oder von einer lokalen CD installiert wird, hängt von der Konfiguration des Programmes *apt-get* ab. Nach einer Systeminstallation von CD ist *apt-get* so konfiguriert, dass Software nur von CD installiert wird. Mit Hilfe des Programmes *apt-setup* lassen sich alternative Installationsquellen wie FTP oder HTTP, aber auch NFS bestimmen.

Der genaue Paketname lässt sich zum einen durch „apt-cache search *suchtext*“ oder durch einen Besuch auf der Seite http://www.debian.org/distrib/packages bestimmen. Auch ist zu beachten, dass die installierbaren Softwareversionen sich je nach gewähltem Entwicklungszweig unterscheiden. Es gibt *stable*, *testing* und *unstable* oder auch *woody*, *sarge* und *sid* genannt. Bei *woody* handelt es sich um das offizielle Release 3, welches für die Zielsetzung dieses Buches zum Teil jedoch nicht ausreichend aktuelle Softwareversionen bietet. Hier ist häufiger eine Softwareversion aus dem *testing* bzw. *sarge* Zweig oder aus dem *unstable* bzw. *sid*-Zweig erforderlich. Aber selbst *sid* genügt den Ansprüchen an Stabilität für diese Zwecke, ohne Konkurenz scheuen zu müssen. Das debian GNU/Linux-Projekt hat sich sehr hohe Maßstäbe gesetzt, bevor ein Softwarepaket in den *stable*-Zweig aufgenommen wird.

apt-setup schreibt in „/etc/apt/sources.list“, eine Konfigurationsdatei, Einträge, die wie folgt aussehen können:

```
# Einträge für den Zugriff auf woody-Sourcen
deb http://ftp.tu-clausthal.de/pub/linux/debian/ stable main non-free contrib
deb-src http://ftp.tu-clausthal.de/pub/linux/debian/ stable main non-free contrib
deb http://non-us.debian.org/debian-non-US stable/non-US main contrib non-free
deb-src http://non-us.debian.org/debian-non-US stable/non-US main contrib non-free

# Einträge für den Zugriff auf sarge-Sourcen
deb     http://ftp.de.debian.org/debian sarge main contrib non-free
deb-src http://ftp.de.debian.org/debian sarge main contrib non-free

# Eintrag für Sicherheitsupdates
deb http://security.debian.org/ stable/updates main contrib non-free
```

Um explizit ein Paket aus *sarge* oder *sid* zu installieren, ist es erforderlich, dem *apt-get install* den Schalter '-t' und den gewünschten Namen des Entwicklungszweiges mitzugeben.

apt-get install -t sarge *paketname*

Selbstverständlich muss eine entsprechende Zeile in „/etc/apt/sources.list“, auf Ressourcen des *sarge*-Zweiges verweisen.

FreeBSD

Deutsches Handbuch

http://www.freebsd.org/doc/de_DE.ISO8859-1/books/handbook/index.html

Englisches Handbuch

http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/index.html

Zum Schwierigkeitsgrad der Systeminstallation gilt für FreeBSD das für debian GNU/Linux Gesagte. Die Installation ist schnell und auch nicht schwerer als andere.

FreeBSD bietet zwei verschiedene native Systeme zur Softwareverwaltung an. Zum einen ein Paketsystem, ähnlich dem von debian GNU/Linux, das vorkompilierte Pakete bereitstellt, welche mit Hilfe einer graphischen Benutzeroberfläche (Aufruf: „/stand/sysinstall“) installiert werden können. Zusätzlich bietet FreeBSD die sogenannte Ports-Sammlung an, bei der es sich um eine thematisch geordnete Sammlung von auf FreeBSD angepassten Makefiles handelt. Die Ports-Sammlung kann bereits während der Systeminstallation installiert werden. Sie befindet sich im Verzeichnis „/usr/ports“. Darunter befinden sich ungefähr weitere 60 Unterverzeichnisse, die die verschiedenen Softwarekategorien repräsentieren. Erst darunter finden sich die eigentlichen Softwareports. Wechselt man nun in eines dieser Verzeichnisse, so kann der entsprechende Port mit Hilfe von *make install* kompiliert und installiert oder mit *make deinstall* deinstalliert werden. Ein *make clean* löscht den heruntergeladenen und kompilierten Quellcode, was nach der Installation der Software gefahrlos gemacht werden kann, sofern dieser nicht, wie im Fall von PostgreSQL und PHP4 im weiteren Verlauf noch benötigt wird.

Als einer der ersten Schritte sollte nach der Systeminstallation die Ports-Sammlung aktualisiert werden. Dazu ist es erforderlich, die Pakete *cvsup-without-gui* und *portupgrade* zu installieren. Auf Systemen mit einem Fenstermanager wie Gnome oder KDE kann auch die graphische Version *cvsup* installiert werden. *cvsup* sollte nicht aus der Ports-Sammlung installiert werden, da *cvsup,* im Gegensatz zu der meisten anderen Software, in der Programmiersprache Modula-3 geschrieben wurde und es erforderlich wäre, diese zuvor auch zu kompilieren und zu installieren, was einige Zeit in Anspruch

nehmen würde. Um die Pakete zu installieren, ruft man von der Kommandozeile „/stand/sysinstall“ auf und wählt den Abschnitt „configure/packages/net“ für *cvsup-without-gui* bzw. den Abschitt „configure/packages/sysutils“ für *portupgrade*.

Um *cvsup* zu verwenden ist eine Konfigurationsdatei, genannt *ports-supfile,* erforderlich. Das folgende Beispiel kann gleich verwendet werden, es muss lediglich das „X“ hinter „cvsup“ durch eine Zahl zwischen 2 und 7 ersetzt werden. Diese steht für einen der deutschen FreeBSD Spiegel-Server:

```
*default host=cvsupX.de.FreeBSD.org
*default prefix=/usr
*default release=cvs tag=.
*default delete use-rel-suffix
ports-all
doc-all
```

Der Aktualisierungsvorgang wird durch „cvsup /Pfad-zum-/ports-supfile” gestartet. Nach Abschluss des Vorgangs müssen der Index, der eine Art Inhaltsverzeichnis für die Ports-Sammlung beinhaltet, und die Readme-Dateien aktualisiert werden, was durch einen Wechsel in das Verzeichnis „/usr/ports“ mit anschließender Eingabe von „make index && make readmes” auf der Kommandozeile erreicht wird. Je nach Leistung des Rechners kann der Vorgang durchaus einige Zeit in Anspruch nehmen. Jedoch ist die Aktualisierung nicht häufig erforderlich. Nach Abschluss der Aktualisierung des Indexes und der Readmes, kann mit Hilfe des Befehls “portupgrade -aR” die installierte Software mit der in der Ports-Sammlung verglichen und veraltete Software durch die neuere ersetzt werden. Sollte der Vorgang einmal fehlschlagen, wird die alte Software wieder installiert, sodass nichts „Schlimmes” passiert ist.

Zwischen jedem Deinstallations- und Installationsvorgang führt 'portupgrade' den Befehl „pkgdb -Fa” aus. Das aktualisiert die Paketverwaltungsdatenbank. Dieser Befehl sollte auch nach jeder manuellen Installation ausgeführt werden.

Weitere wichtige Kommandos:

'pkg_info [|grep Paketname/Teil eines Paketnames]' listet [alle] installierte Pakete auf
'portupgrade <portname>' aktualisiert ein bestimmtes Paket
'portupgrade -aR' aktualisiert alle installierten Pakete

Beide Systeme installieren automatisch Pakete bzw. Ports mit, von welchem die zu installierende Software abhängt.

Sehr ähnliche Systeme bieten auch OpenBSD und NetBSD an. Jedoch bietet FreeBSD die größte Auswahl an Paketen und Ports an.

FreeBSD bietet zum einen auch Linux Binär-Kompatibilität, sodass Linux-Software installiert werden kann und es ist sogar möglich RPM-Pakete zu installieren, sofern der dafür erforderliche *rpm* Port installiert ist.

Um den genauen Namen eines Pakets bzw. Ports herauszufinden ist ein Besuch der Seite http://www.freebsd.org/ports/index.html die sicherlich komfortabelste Methode. Weitere Suchoptionen finden sich im graphischen Frontend oder auf der Kommandozeile mit z.B. 'make search name=apache' (im Verzeichnis /usr/ports ausgeführt!). Letzteres benötigt den zuvor aktualisierten Index der Portssammlung.

Quellcode kompilieren

Auf allen Systemen ist es möglich, sofern ein Kompilierer installiert ist, Quellcode zu kompilieren und diesen zu installieren. Das ist dann erforderlich, wenn eine Distribution nicht die benötigte Softwareversion als Paket anbietet oder die benötigte Software überhaupt nicht als Paket vorhanden ist.

Zuerst muss das gewünschte Archiv (häufig ein *tarball*, mit der Erweiterung *tar.gz* oder *tgz*) heruntergeladen und mit Hilfe von „tar -xzf *archivname“* entpackt werden. Die meisten Softwarepakete enthalten ein configure-Skript, welches mit *./configure [Optionen]* (der vorangestellte Punkt ist erforderlich!) aufgerufen wird. Hiernach erfolgt ein *make*, das den Quellcode kompiliert, und ein *make install*, wodurch die soeben kompilierte Software installiert wird. Es folgt ein *ldconfig* (unter Linux. Achtung: bei FreeBSD löscht *ldconfig* ohne weitere Schalter sämtliche Einträge!), um den Cache für die Pfade zu den Bibliotheken zu aktualisieren. Einige Pakete bieten zur Deinstallation auch ein *make uninstall* an.

```
#./configure
# make
# make install
# ldconfig
```

Zu beachten ist, dass Softwarepakete, von denen die zu installierende Software abhängig ist, nicht automatisch mit installiert werden. Falls eine Abhängigkeit nicht erfüllt ist, bricht entweder das configure-Skript oder spätestens die Kompilierung mit einer mehr oder weniger sprechenden Fehlermeldung ab.

2.3 Zu installierende Pakete

- PostgreSQL

Version 7.3 ist gegenüber Version 7.2 der Vorzug zu geben. Version 7.4 ist für PostGIS 0.8.1 und UMN-MapServer 4 freigegeben.

Zur Installation von PostGIS, auf die gegen Ende dieses Kapitels eingegangen wird, ist ein

kompilierter Quellbaum von PostgreSQL erforderlich. Hier kann entweder auf ein src-RPM bzw. src-deb Paket oder auf den Original-Quellcode zurückgegriffen werden, sofern es sich hierbei um dieselbe Version handelt.

Falls von einem entfernten Rechner auf die PostgreSQL Datenbank zugegriffen werden soll, ist in der Datei *postgresql.conf* der Wert für „tcpip_socket" auf „true" zu setzen und das Kommentarzeichen (#) zu entfernen, sowie in der Datei *pg_hba.conf* ein Eintrag hinzuzufügen, der entweder den Host oder das Netzwerk enthält, welches auf die PostgreSQL Datenbank zugreifen können soll.

Installationsanweisungen

Die Standardpartitionen sind unter Umständen nicht ausreichend groß, um größere Datenbanken aufzunehmen. Auch ist eine Separierung dieser Daten empfehlenswert. Daher ist es ratsam, die Datenbank-Dateien an einen anderen Ort zu verschieben. Sehr schnell wird das erreicht, indem ein symbolischer Link gesetzt wird, z.B„ln -s / usr/local/pgsql/data/data/pgsql". Hierzu muss der Datenbankdienst zuvor angehalten werden (FreeBSD: „/usr/local/etc/rc.d/010.pgsql.sh stop", debian/SuSE: „/etc/init.d/posgresql stop")!

FreeBSD

Verzeichnis: /usr/ports/databases/postgresql73

Kommando: *make install && pkgdb -Fa*

make clean sollte nicht benutzt werden, da der Quellbaum zur späteren Installation von PostGIS noch erforderlich ist.

Unter FreeBSD ist eine manuelle Initialisierung der Datenbank erforderlich.

```
# mkdir /usr/local/pgsql/data (FreeBSD Standardverzeichnis für pgSQL)
# chown pgsql /usr/local/pgsql/data
# su - pgsql
# /usr/local/bin/initdb -D /usr/local/pgsql/data
# /usr/local/bin/postmaster -D /usr/local/pgsql/data
```

debian GNU/Linux

Zur Installation von PostgreSQL 7.3 muss auf die Version aus dem Entwicklungszweig sarge/testing zurückgegriffen werden. Das „-dev" Paket wird benötigt. Der Quellbaum ist zur späteren Installation von PostGIS erforderlich.

apt-get install -t sarge postgresql-dev

Anmerkung:
Da ich nicht herausfinden konnte, wie der aus sarge installierte Quellbaum kompiliert wird, habe ich auf die originalen Sourcen von postgresql.org zurückgegriffen.

SuSE Linux

Yast: postgresql-devel-7.3.4

Yast: postgresql-server-7.3.4

rpm -i postgresql-7.3.4-53.src.rpm

Das src-Paket enthält den Quellcode. Er wird unter "/usr/src/packages/SOURCES" "installiert" und muss anschließend entpackt und kompiliert werden:

```
# tar -xjf postgresql-7.3.4.tar.bz2
# cd postgresql-7.3.4/
# ./configure –without-readline (falls readline-Support nicht installiert ist)
# make
```

- MySQL

MySQL wird für die Client Suite Mapbender benötigt.

Installationsanweisungen

FreeBSD

Verzeichnis: */usr/ports/databases/mysql40-server*

Kommando: *make install clean && pkgdb –Fa*

debian GNU/Linux

Das devel-Paket ist erforderlich, um PHP4 mit „--with-mysql" kompilieren zu können. MySQL Version 4 ist erst im Testing-Zweig „sarge" enthalten.

apt-get install -t sarge mysql-dev

SuSE Linux

Das devel-Paket ist erforderlich, um php mit „--with-mysql" kompilieren zu können.

Yast: mysql-devel-4.0.15

Yast: mysql-client-4.0.15

- Apache

Da die Zusammenarbeit von Apache 2 mit der Client Suite Mapbender nicht reibungslos funktioniert, greifen wir nach wie vor auf Apache 1.3 zurück.

In der Konfigurationsdatei des Apache Webservers muss der „ServerName" angegeben werden. Auf einem Entwicklungsrechner, der nicht im Internet steht, kann das auch „localhost" oder „127.0.0.1" sein. Andernfalls sollte hier der FQDN (Full qualified Domain Name) stehen. Auch sollte hierfür ein Eintrag in der Datei „/etc/hosts" vorhanden sein!

Installationsanweisungen

FreeBSD

Verzeichnis: */usr/ports/www/apache13*

Kommando: *make install clean && pkgdb –Fa*

debian GNU/Linux

apt-get install apache

SuSE Linux

Yast: apache-1.3.28

- PHP4

Es kann aus zweierlei Gründen erforderlich sein, PHP4 zu installieren. Zum einen bietet der UMN-MapServer mit phpMapScript erweiterte Möglichkeiten, und zum anderen wird PHP4 für die Client Suite Mapbender benötigt.

PhpMapScript

Grundsätzlich ist es aus Sicherheitsgründen empfehlenswert, PHP4 als Apache Modul zu

installieren. Soll jedoch phpMapScript zum Einsatz kommen, muss PHP4 als cgi kompiliert werden. In diesem Fall sollte zusätzlich die Installation von „suexec" oder „suphp" (www.suphp.org) in Betracht gezogen werden!

(Siehe http://www.php.net/manual/en/security.cgi-bin.php)

PHP4 wird als cgi kompiliert, indem die configure-Option „--with-apxs" nicht angegeben wird. Außerdem muss PHP4 mit der Option „--with-regex=system" kompiliert werden. Da keine der (mir bekannten) Linux/Unix Distributionen ein mit „--with-regex=system" vorkompiliertes PHP4-cgi Paket anbieten, ist die Kompilierung von der Quelle auf den hier genannten Linux Distributionen erforderlich. Zwar bietet debian GNU/Linux ein php4-cgi Paket an, jedoch ist das nicht mit der Option „--with-regex=system" kompiliert worden. FreeBSD bietet hier am meisten Spielraum, da das Makefile aus dem Ports-Tree dahingehend geändert werden kann.

Weitere Information:

http://mapserver.gis.umn.edu/data2/wilma/mapserver-users/0208/msg00354.html

http://mapserver.gis.umn.edu/cgi-bin/wiki.pl?PHPMapScript

http://mapserver.gis.umn.edu/cgi-bin/wiki.pl?PHPMapScriptCGI

Client Suite Mapbender

Für die Verwendung der Client Suite Mapbender kann PHP4 sowohl als Apache Modul, als auch als cgi installiert werden. Eine Kompilierung von PHP4 auf SuSE Linux, Red Hat, sowie Mandrake ist unumgänglich, da das Paket „mod_php4" nicht mit „domxml"-Unterstützung kompiliert wurde und es im Gegensatz zu debian GNU/Linux nicht im Nachhinein als Modul installiert werden kann.

Erforderliche Optionen für PHP4

```
--enable-dbase
--enable-force-cgi-redirect
--with-jpeg-dir[=DIR]
--with-mysql[=DIR]
--with-pgsql[=DIR]
--with-png-dir[=DIR]
--with-tiff-dir[=DIR]
--with-freetype-dir[=DIR]
--with-xml
--with-zlib=yes
--with-gd
```

--with-curl[=DIR]
--with-regex=system
--with-dom[=DIR] (Mapbender Client Suite)

***Installationsanweisungen (nur Mapbender, ohne phpMapScript Unterstützung*)**

Falls nach der Installation das PHP4 Modul nicht automatisch geladen wurde, kann das auf allen Systemen durch „apachectl graceful" erreicht werden.

FreeBSD
In einem Dialogfenster wird der Benutzer nach Ausführen von *make* zur Angabe der gewünschten Optionen aufgefordert.

Verzeichnis: */usr/ports/www/mod_php4*
Kommando: *make install clean && pkgdb –Fa*
debian GNU/Linux
apt-get install -t sid php4
apt-get install -t sid php4-pgsql, php4-mysql, php4-dbase, php4-domxml, php4-gd2, php4-curl
SuSE Linux
Kompilierung des Quellcodes ist erforderlich. Kompilierungsanweisungen folgen im Abschnitt „2. Für phpMapScript-Unterstützung" unter SuSE Linux.

Installationsanweisungen (mit phpMapScript Unterstützung)

FreeBSD
Um die empfohlenen Optionen in PHP4 mit einzukompilieren ist es erforderlich im Makefile von PHP4 („/usr/ports/lang/php/") die Zeile 'CONFIGURE_ARGS+=--enable-discard-path' auszukommentieren, da es andernfalls zu folgenden Fehlern kommt:

1. *Browser: parse error*
2. *httpd-error-log: Premature end of script header*

Hierbei scheint es sich um einen Bug in php zu handeln.

Nun kann der Installationsvorgang beginnen. In einem Dialogfenster wird der Benutzer zur Angabe der gewünschten Optionen aufgefordert.

Verzeichnis: */usr/ports/www/php4-cgi*

Kommando: *make WITH_REGEX_TYPE="system" && make install && pkgdb -Fa*

Da der Quellcode später noch benötigt wird, um UMN-MapServer mit der Option „--with-php4=path-to-source" zu kompilieren, wird *make clean* an dieser Stelle nicht ausgeführt!

Nach Abschluss der Installation muss das PHP4-Binary (/usr/local/bin/php) in das „cgi-bin" Verzeichnis kopiert werden (FreeBSD Standard: /usr/local/www/cgi-bin) und die beiden folgenden Zeilen in die Konfigurationsdatei *httpd.conf* des Apache-Webservers unterhalb von *AddType application/x-tar .tgz* eingefügt werden:

```
AddType application/x-httpd-php .php
Action application/x-httpd-php /cgi-bin/php
```

debian GNU/Linux
apt-get install php4-cgi (phpMapScript ohne mapbender)
apt-get source php4-cgi (phpMapScript ohne mapbender)
apt-get install -t sarge php4-cgi (mit mapbender und phpMapScript)
apt-get source -t sarge php4-cgi (mit mapbender und phpMapScript)

Mit „apt-get source" wird der zum Paket passende Quellcode in das Verzeichnis, in welchem man sich gerade befindet, heruntergeladen. Normalerweise handelt es sich um ein tarball-Archiv, welches nun noch entpackt und kompiliert werden muss.

1. tar -xzvf php4-archiv.tgz
2. cd php4-archiv
3. ./configure

Die Optionen sind soweit wie möglich den Optionen des Original-Pakets entnommen. Hinzu kommen die, die für phpMapScript bzw. Mapbender erforderlich sind. Die, die nicht funktionierten und auch nicht zwingend erforderlich sind, wurden entfernt.

```
./configure --enable-force-cgi-redirect –with-regex=system --with-config-file-path=/etc/php4/cgi --prefix=/usr --with-pear=/usr/share/php  --enable-memory-limit --enable-ctype --enable-sysvsem --enable-sysvshm --enable-track-vars --enable-trans-sid --enable-sockets --disable-static --disable-debug --disable-rpath --enable-calendar --enable-bcmath --with-iconv --enable-exif --enable-filepro --enable-ftp --with-gettext --enable-mbstring --enable-shmop --enable-wddx --with-expat-dir=/usr --enable-yp --with-zlib --with-xml=/usr --with-freetype-dir=/usr/lib --with-jpeg-dir=/usr/lib --with-png-dir=/usr/lib --with-tiff-dir=/usr/lib --without-sysbase-ct --without-pgsql --without-mm --with-mysql --with-kerberos=/usr --with-openssl=/usr --with-exec-dir=/usr/lib/php4/libexec --enable-dbase –with-gd
```

4. make
5. mv /usr/lib/cgi-bin/php /usr/lib/cgi-bin/php.original

Sicherung des Original php-Binaries

6. cp ../php-source-tree/php /usr/lib/cgi-bin

Das neu kompilierte Binary wird in das cgi-bin Verzeichnis kopiert.

7. Anpassung der Apache Konfigurationsdatei httpd.conf für PHP4. Zu positionieren unterhalb von *AddType application/x-tar .tgz.*

AddType application/x-httpd-php .php
Action application/x-httpd-php /cgi-bin/php

Und als letzter Schritt, einkommentieren der Zeile
LoadModule action_module /usr/lib/apache/1.3/mod_actions.so

SuSE Linux
Wegen der verwendeten configure-Optionen wird folgendes RPM-Paket benötigt:

Yast: flex-2.5.4a

1. tar -xzvf php4-archiv.tgz
2. cd php4-archiv
3. ./configure:

```
# ./configure --prefix=/usr/share --datadir=/usr/share/php --bindir=/usr/bin --libdir=/usr/share --includedir=/usr/include --with-_lib=lib --with-config-file-path=/etc --with-exec-dir=/usr/lib/php/bin --disable-debug --enable-bcmath --enable-calendar --enable-ctype --enable-dbase --enable-exif --enable-filepro --enable-force-cgi-redirect --enable-ftp --enable-gd-imgstrttf --enable-gd-native-ttf --enable-inline-optimization --enable-magic-quotes --enable-mbstr-enc-trans --enable-mbstring --enable-mbregex --enable-memory-limit --enable-safe-mode --enable-shmop --enable-sigchild --enable-sysvsem --enable-sysvshm --enable-track-vars --enable-trans-sid --enable-versioning --enable-wddx --enable-yp --with-bz2 --with-dom=/usr/include/libxml2 --with-gettext --with-jpeg-dir=/usr --with-mysql=/usr --with-pgsql=/usr --with-png-dir=/usr --with-tiff-dir=/usr --with-ttf --with-freetype-dir=yes --with-xml --with-zlib=yes --with-gd  --with-curl=/usr/local --with-iconv --with-regex=system
```

4. make
5. make install
6. cp ../php-source-tree/sapi/cgi /srv/www/cgi-bin/

Das neu kompilierte Binary wird in das „cgi-bin”-Verzeichnis kopiert.

7. Anpassung der Apache Konfigurationsdatei *httpd.conf* für PHP4. Zu positionieren unterhalb von *AddType application/x-tar .tgz.*

   AddType application/x-httpd-php .php
   Action application/x-httpd-php /cgi-bin/php

Falls sich in „/etc” keine *php.ini* befindet, muss diese aus „/path/to/src-code/php.ini-dist” dorthin kopiert und entsprechend umbenannt werden.

Letzte Schritte (für alle Systeme)

Ein Verzeichnis Alias für den Apache Webserver muss eingerichtet werden, um dem UMN MapServer Web-Zugriff zu demm Verzeichnis, welches im map-file mit „IMAGEURL” referenziert wird, zu erlauben, z.B:

```
Alias /umn/ "/data/umn/umn-www"

<Verzeichnis "/data/umn/umn-www">
        Options All
        AllowOverride None
        Order allow,deny
        Allow from all
</Verzeichnis>
```

Bibliotheken

Auf den Linux Systemen ist die Installation der -dev (debian GNU/Linux) bzw. -devel (SuSE Linux, etc) Pakete erforderlich, damit die zur Kompilierung des UMN-MapServer und von PHP4 erforderlichen Header-Dateien auf dem System vorhanden sind.

http://mapserver.gis.umn.edu/doc36/unix-install-howto.html#obtain
http://mapserver.gis.umn.edu/doc40/unix-install-howto.html#obtain

- gd

FreeBSD
Die gd2 sollte an dieser Stelle bereits als Abhängigkeit durch die PHP4-Installation auf dem System vorhanden sein.

Verzeichnis: */usr/ports/graphics/gd2*
Kommando: *make install clean && pkgdb –Fa*
debian GNU/Linux
apt-get install -t sarge libgd2-dev
SuSE Linux
Yast: gd-devel-2.0.15

- freetype2-2.1.4

Sollte als Abhängigkeit von gd2 bereits installiert sein.

FreeBSD
Verzeichnis: */usr/ports/print/freetype2*
Kommando: *make install clean && pkgdb –Fa*
debian GNU/Linux
libfreetype6-dev muss aus dem gleichen debian GNU/Linux Entwicklungszweig wie die zuvor installierte php4-gd2 installiert werden!

apt-get install [-t sarge] libfreetype6-dev
SuSE Linux
Yast: freetype2-devel-2.1.4

- jpeg-6b

Sollte als Abhängigkeit von gd2 bereits installiert sein.

FreeBSD
Verzeichnis: */usr/ports/graphics/jpeg*
Kommando: *make install clean && pkgdb –Fa*
debian GNU/Linux
apt-get install libjpeg62-dev
SuSE Linux
Ein „devel-Paket“ gibt es nicht.

Yast: libjpeg-6.2.0

- png-1.2.5

Sollte als Abhängigkeit von gd2 bereits installiert sein.

FreeBSD
Verzeichnis: */usr/ports/graphics/png*
Kommando: *make install clean && pkgdb –Fa*
debian GNU/Linux
apt-get install libpng2-dev
SuSE Linux
Yast: libpng-devel-1.2.5

- gdal-1.1.8/1.1.9

FreeBSD
Verzeichnis: */usr/ports/graphics/gdal*
Kommando: *make install clean && pkgdb –Fa*
debian GNU/Linux
Ein Beta Paket der Version 1.1.9 steht auf
http://mentors.debian.net/debian/dists/unstable/main/binary-i386/gdal/
zur Verfügung. Es gibt außerdem ein „libgdal1-dev” Paket, Version 1.2, im sarge/testing Zweig.

Anmerkung:
Bisher haben wir gdal aus dem Quellcode kompiliert und installiert. Erfahrungen mit den oben beschriebenen Paketen haben wir noch keine gemacht.

SuSE Linux
Da SuSE und Red Hat kein eigenes RPM-Paket anbieten, ist die Kompilierung und Installation des Quellcodes erforderlich.

Für beide Linux-Derivate gilt

Sollte der UMN-MapServer mit der entsprechenden Fehlermeldung (Bibliothek nicht gefunden) nicht starten, so sollte folgendes noch geändert werden:

```
# vi /etc/ld.so.conf (einfügen /usr/local/lib)
# ldconfig –v (Überprüfen, ob der Eintrag vorhanden ist)
```

- proj-4.4.5

FreeBSD
Verzeichnis: /usr/ports/graphics/proj
Kommando: make install clean && pkgdb –Fa
debian GNU/Linux
apt-get install proj
SuSE Linux
Da SuSE und Red Hat kein eigenes RPM-Paket anbieten, ist die Kompilierung und Installation des Quellcodes erforderlich.
Für alle Systeme
Auf manchen Systemen scheint UMN-MapServer den Namen der Datei 'epsg' in Grossbuchstaben zu erwarten. Bisher haben wir immer einen symbolischen Link in „/usr/local/share/proj" erstellt.

```
# ln –s epsg EPSG
```

- GEOS

Dieser Port ist erst in der Ports-Sammlung **nach** einer Aktualisierung mit cvsup enthalten.

FreeBSD
Verzeichnis: */usr/ports/graphics/geos*
Kommando: *make install clean && pkgdb –Fa*
debian GNU/Linux
debian GNU/Linux bietet kein eigenes deb-Paket an, deshalb ist die Kompilierung von der Quelle erforderlich.
SuSE Linux
SuSE bietet kein eigenes RPM-Paket an, deshalb ist die Kompilierung von der Quelle erforderlich.

- libwww-5.4.0

Nur für UMN-MapServer 3.6 erforderlich.
FreeBSD
Verzeichnis: */usr/ports/www/libww*
Kommando: *make install clean && pkgdb –Fa*
debian GNU/Linux
apt-get install libwww-dev
SuSE Linux
SuSE bietet kein eigenes RPM-Paket an, deshalb ist die Kompilierung von der Quelle erforderlich.

- curl-7.10

Für UMN-MapServer 4 erforderlich.

FreeBSD
Verzeichnis: */usr/ports/ftp/curl*
Kommando: *make install clean && pkgdb –Fa*
debian GNU/Linux
apt-get install –t sarge libcurl-dev
SuSE Linux
Die curl-Version, die auf SuSE Linux 8.2 zur Verfügung steht, ist nicht aktuell genug, weshalb die Kompilierung von der Quelle erforderlich ist.

- libiconv-1.9.1

Für UMN-MapServer 4 erforderlich.
FreeBSD
Verzeichnis: */usr/ports/converters/libiconv*
Kommando: *make install clean && pkgdb –Fa*
debian GNU/Linux
apt-get install -t sarge libiconv-hook-dev
SuSE Linux
SuSE/Red Hat bieten kein eigenes RPM-Paket an, deshalb ist die Kompilierung von der Quelle erforderlich.

- tiff-3.5.7

FreeBSD
Verzeichnis: */usr/ports/graphics/tiff*
Kommando: *make install clean && pkgdb –Fa*
debian GNU/Linux
apt-get install libtiff3g-dev

SuSE Linux
Ein „devel-Paket“ gibt es nicht.

Yast: libtiff-3.5.7

- Libxml2

FreeBSD
Wird als Abhängigkeit bei der PHP Installation mitinstalliert.

Verzeichnis: */usr/ports/textproc/libxml2*
Kommando: *make install clean && pkgdb –Fa*
debian GNU/Linux
Wird als Abhängigkeit bei der php-xml-Modul Installation mitinstalliert.

SuSE Linux
Yast: libxml2-devel-2.5.10

- Yacc

Auf SuSE Systemen ist Yacc möglicherweise für die Kompilierung von UMN-MapServer erforderlich, aber auch meistens bereits installiert.

- xerces-1.6.0

FreeBSD
Sofern gml-Unterstützung nicht benötigt wird, ist eine Installation von xerces nicht erforderlich.

Die unter FreeBSD verfügbare Version von gdal 1.1.8 erkennt nur die xerces Version 1.6.0. Diese jedoch lässt sich auf einem aktuellen FreeBSD nicht aus den Ports installieren. Hier ist „nur“ Version xerces-2.3 verfügbar. Allerdings ist dieser Bug dem Maintainer der gdal bekannt und in der Bearbeitung. Deshalb müsste bis dahin auf den Original-Quellcode zurückgegriffen werden.

debian GNU/Linux
Wurde noch nicht getestet.

SuSE Linux
Wurde noch nicht getestet.

PostGIS 0.7.5/0.8.1

Postgis muss im Verzeichnis *contrib* des PostgreSQL Quellbaumes entpackt werden.

```
# cd /Pfad/zu/pgsql/Quellbaum/contrib
# tar -xzvf postgis-0.7.5.tar.gz
```

Nun muss entweder das soeben entstandene Verzeichnis *postgis-0.7.5* in *postgis* umbenannt werden

```
# mv postgis-0.7.5 postgis
```

oder der entsprechende Eintrag im Makefile von PostGIS geändert werden „subdir=contrib/postgis”. Außerdem kann der Standardwert für „USE_PROJ” übernommen werden. Steht er auf „1” (PostGIS 0.8 Standard), wird die *proj4*-Bibliothek benötigt, die aber ohnehin bereits installiert worden sein sollte. Da jedoch der UMN-MapServer „On-the-fly” Umprojizierung beherrscht, kann der Wert auch auf „0” belassen bzw. geändert werden. Seit PostGIS 0.8 kann GEOS Funktionalität mit eingebunden werden. Der Standardwert für „USE_GEOS” ist „1”, deshalb kann dieser belassen werden, sofern GEOS-Unterstützung gewünscht und eine Installation von GEOS auf dem System vorhanden ist. Nun kann PostGIS kompiliert werden. Unter **Linux** mit **make**, unter **FreeBSD** mit **gmake**!

```
# gmake
```

```
# gmake install
```

PostGIS Funktionen in PostgreSQL laden

Die Datei *postgis.sql* verwendet die Variable $libdir. Diese Variable enthält den Pfad zu den PostGIS Bibliotheken. Auf manchen Systemen wurde der Wert für $libdir falsch belegt und das Ausführen der Datei *postgis.sql* führt zu einer Menge von Fehlermeldungen.

Um das zu beheben, kann zum einen in der Datei *postgis.sql* die Variable '$libdir' durch den absoluten Pfad, welcher mit Hilfe von „pg_config --pkglibdir" zu ermitteln ist, zu den entsprechenden Bibliotheken ersetzt werden, oder die Bibliotheken an die erwartete Stelle kopiert werden.

FreeBSD

Die betreffenden Bibliotheken befinden sich in„/usr/local/lib/".
„pg_config --pkglibdir" zeigt „/usr/local/pgsql/lib" an.

```
# cp /usr/local/lib/libpostgis.a /usr/local/pgsql/lib
# cp /usr/local/lib/libpostgis.so.0 /usr/local/pgsql/lib
```

In „pg_config --pkglibdir" (/usr/local/pgsql/lib) muss ein symbolischer Link erstellt werden.

```
# ln –s libpostgis.so.0 libpostgis.so
```

Mit PostGIS 0.8 ist dieses Prozedere nicht erforderlich.

Debian GNU/Linux

Die betreffenden Bibliotheken befinden sich in „/usr/local/pgsql/lib/".
„pg_config --pkglibdir" zeigt „/usr/lib/postgresql/lib" an.

```
# cp /usr/local/pgsql/lib/libpostgis.a /usr/lib/postgresql/lib
# cp /usr/local/pgsql/lib/libpostgis.so.0.7 /usr/lib/postgresql/lib
```

In „pg_config --pkglibdir" (/usr/local/pgsql/lib) muss ein symbolischer Link erstellt werden.

```
# ln –s libpostgis.so.0.7 libpostgis.so
# ln –s libpostgis.so.0.7 libpostgis.so.0
```

Mit PostGIS 0.8 unter debian GNU/Linux liegen uns noch keine Erfahrungen vor.

SuSE Linux

Die betreffenden Bibliotheken befinden sich in „/usr/local/pgsql/lib/".
„pg_config --pkglibdir" zeigt „/usr/lib/postgresql" an.

```
# cp /usr/local/pgsql/lib/libpostgis.a /usr/lib/postgresql (PostGIS 0.7)
```

cp /usr/local/pgsql/lib/libpostgis.so.0.7 /usr/lib/postgresql (PostGIS 0.7)

cp /usr/local/pgsql/lib/libpostgis.so.0.8 /usr/lib/postgresql (PostGIS 0.8)

In „pg_config --pkglibdir" (/usr/lib/postgresql) müssen zwei symbolische Links erstellt werden.

ln –s libpostgis.so.0.7 libpostgis.so (PostGIS 0.7)
ln –s libpostgis.so.0.7 libpostgis.so.0 (PostGIS 0.7)

ln -s libpostgis.so.0.8 libpostgis.so (PostGIS 0.8)
ln -s libpostgis.so.0.8 libpostgis.so.0 (PostGIS 0.8)

Letzte Schritte
Die folgenden Schritte sind erforderlich, um PostGIS-Unterstützung in eine zuvor zu erstellende Datenbank zu installieren (http://postgis.refractions.net/docs/x83.html).

pgSQL-Kommando, zur Erstellung einer neuen Datenbank
createdb *datenbankname*

PostGIS benötigt die prozedurale Spracherweiterung PL/pgSQL procedural language extension
createlang plpgsql *datenbankname* (Es erfolgt *keine* Bestätigung!)

Die beiden SQL-Dateien befinden sich unter FreeBSD in „/usr/local/share/postgresql/contrib/", unter SuSE Linux und debian GNU/Linux in „/usr/local/pgsql/share/contrib/" oder einfach im Quellbaum von PostGIS.

Nun können die PostGIS Objekt- und Funktionsdefinitionen in die zuvor erstellte Datenbank geladen werden.
psql -d *datenbankname* -f postgis.sql

Um eine vollständige Auflistung der EPSG Koordinatensystem Identifier zu erhalten, kann die *spatial_ref_sys.sql* Definitionsdatei geladen werden und die SPATIAL_REF_SYS Tabelle befüllt werden.
psql -d *datenbankname* -f spatial_ref_sys.sql

Die folgende Ausgabe ist nach Ausführen von *postgis.sql* gleich zu Beginn zu sehen. Dabei handelt es sich nicht um einen Fehler, siekann deshalb ignoriert werden!

```
> psql -d test2 -f /usr/ports/databases/postgresql73/postgresql-7.3.5/contrib/postgis-0.8.1/postgis.sql
psql:/root/postgresql/postgresql-7.3.5/contrib/postgis-0.8.1/postgis.sql:18: NOTICE:  ProcedureCreate: type
histogram2d is not yet defined
psql:/root/postgresql/postgresql-7.3.5/contrib/postgis-0.8.1/postgis.sql:23: NOTICE:  Argument type
"histogram2d" is only a shell
.
```

.
psql:/root/postgresql/postgresql-7.3.5/contrib/postgis-0.8.1/postgis.sql:154: NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index 'spatial_ref_sys_pk
ey' for table 'spatial_ref_sys'
psql:/root/postgresql/postgresql-7.3.5/contrib/postgis-0.8.1/postgis.sql:174: NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index 'geometry_columns_p
k' for table 'geometry_columns'

UMN MapServer

Allgemeines

Das Archiv, welches den Quellcode enthält, muss entpackt werden:

 # tar -xzvf umn-mapserver-4.0.1.tar.gz.

Der Wert für MS_MAXSYMBOLS in der Datei *mapsymbol.h* kann erhöht werden, um mehr Symbolklassen zu unterstützen, z.B.: # define MS_MAXSYMBOLS 250

Der Pfad hinter „--with-php=[DIR]" zeigt auf das Verzeichnis, in welchem sich der Quellbaum von PHP4 befindet. Diese Option muss nur angegeben werden, sofern phpMapScript Unterstützung benötigt wird.

Bevor der Quellcode kompiliert werden kann, muss das "configure"-Skript ausgeführt werden, welches sich im entpackten Verzeichnis des Quellcodes befindet.

cd umn-mapserver-4.0.1
./configure [Optionen – siehe unten!]

UMN-MapServer 3.6

Um mehr als 64 Klassen zu unterstützen, muss die Datei *map.h* im Quellbaum von UMN-MapServer bearbeitet werden und dort der Standardwert MS_MAXCLASSES=64 erhöht werden, z.B.: #define MS_MAXCLASSES 250

FreeBSD

./configure –with-gd –with-gdal --with-ogr –with-proj=/usr/local --with-jpeg=/usr/local --with-png=/usr/local --without-tiff --with-zlib --with-freetype --with-php=/usr/ports/www/php4-cgi/work/php-4.3.4 --with-wmsclient --with-postgis --enable-runpat

Anmerkung:
Unter FreeBSD habe ich UMN-MapServer nicht „--with-pdf" kompilieren können.

Zur Unterstützung von phpMapScript unter FreeBSD muss das Makefile an mehreren Stellen angepasst werden (../mapserversource/mapscript/php3/Makefile). Unter Linux ist das nicht erforderlich.

CC= cc zu CC= gcc
LD= cc -Wl -Wl, zu LD= gcc -Wl -Wl,-share

-I$(PHP_SRC_DIR)/dl zu -I$(PHP_SRC_DIR)/ext

Das ist der Ausschnitt aus dem zu bearbeitenden Teil des Makefiles:

```
CC =     gcc    -I. -I.. -DPIC -fpic
LD =     gcc -Wl -Wl,-share
CFLAGS = -O2  -Wall -DCOMPILE_DL=1 -DPHP4
RUNPATHS=  -R/usr/local/lib -R/usr/home/src/mapserver-3.6.5

#
# Set PHP_SRC_DIR to point to the root of the PHP source tree
#
PHP_SRC_DIR = /usr/ports/www/mod_php4/work/php-4.3.1

PHP_INC = -I$(PHP_SRC_DIR) -I$(PHP_SRC_DIR)/ext -I$(PHP_SRC_DIR)/main \
        -I$(PHP_SRC_DIR)/Zend -I$(PHP_SRC_DIR)/include \
        -I$(PHP_SRC_DIR)/TSRM
```

Bei der Verwendung von PosgreSQL 7.3 ist eine geänderte *mappostgis.c* erforderlich. Diese kann hier http://mapserver.gis.umn.edu/data2/wilma/mapserver-users/0304/msg00305.html einfach per „Copy and Paste" übernommen werden.

debian
./configure --with-gd --with-gdal --with-ogr --with-proj --with-jpeg=/usr/ --with-png=/usr/ --without-tiff --with-zlib --with-freetype --with-php=/var/src/php4-4.3.2+rc3 --with-wmsclient --with-postgis

SuSE Linux
./configure --with-gd=/usr --with-gdal --with-ogr --with-proj=/usr/local --with-wmsclient --without-tiff --with-php=/data/src/php-4.3.3 --with-postgis

UMN MapServer 4.0

FreeBSD
./configure --with-gd --with-gdal --with-ogr --with-proj=/usr/local --with-jpeg=/usr/local --with-pdf=/usr/local --with-png=/usr/local --without-tiff --with-libiconv=/usr/local --with-zlib --with-curl --with-freetype --with-php=/usr/ports/www/php4-cgi/work/php-4.3.3 --with-wmsclient --with-wfs --with-wfsclient --with-postgis --enable-runpat

debian GNU/Linux

./configure --with-gd --with-gdal --with-ogr --with-proj=/usr --with-jpeg=/usr --with-png=/usr --without-tiff --with-libiconv=/usr --with-zlib=/usr --with-freetype --with-curl=/usr --with-php=/var/src/php4-4.3.2+rc3 --with-wmsclient --with-wfs --with-wfsclient --with-postgis

SuSE Linux

./configure --with-proj --with-wmsclient --with-wfs --with-wfsclient --with-postgis --with-ogr --without-tiff --with-php=/data/src/php-4.3.4 --with-gdal

Das Finale

Nun muss der Quellcode noch mit 'make' (UMN-MapServer 4 unter FreeBSD mit **gmake**!) kompiliert werden...

Achtung NICHT 'make install' verwenden!

... und das kompilierte Binary 'mapserv' in das cgi-Verzeichnis kopiert werden.

Default cgi-bin Verzeichnis FreeBSD: /usr/local/www/cgi-bin
Default cgi-bin Verzeichnis debian GNU/Linux: /usr/lib/cgi-bin
Default cgi-bin Verzeichnis SuSE Linux: /srv/www/cgi-bin

Windows

Installation eines Webservers

Apache

Der Apache Webserver wird zum Beispiel als „apache_1.3.29-win32-x86-no_src.exe“ oder „apache_1.3.29-win32-x86-src.msi“ heruntergeladen und kann als Windows Installer auf gewohnte Weise installiert werden.

Während der Installation wird der Benutzer zur Eingabe folgender Informationen aufgefordert:

> Domäne
> Name des Servers
> --> Beides ist aus „Systemsteuerung >> System >> Netzwerkidentifikation“ abzulesen.
>
> Email Adresse des Webadministrators
> --> Hier kann auf einem Entwicklungsrechner selbstverständlich eine Dummy-Adresse angegeben werden. Andernfalls sollte man die Adresse des Webmasters angeben.

Außerdem besteht die Möglichkeit, während der Installation festzulegen, ob der Apache Webserver als Dienst läuft. In den meisten Fällen ist das die sinnvollste Wahl, da auch der Dienst auf Wunsch leicht über die Windows Verwaltungsoberfläche abgeschaltet werden

kann.

Konfiguration für den Einsatz mit UMN-MapServer

Der Konfigurationsdatei *httpd.conf* müssen folgende drei Zeilen an passender Stelle hinzugefügt werden. Am Ende der *LoadModule* und des *AddModule*-Blocks die beiden folgenden Zeilen

```
LoadModule php4_module C:\PHP\sapi\php4apache.dll
AddModule mod_php4.c
```

und unterhalb von *AddType application/x-tar .tgz*

```
AddType application/x-httpd-php .php
```

Ein Verzeichnis Alias für den Apache Webserver muss eingerichtet werden, um dem UMN-MapServer Web Zugriff zum Verzeichnis, welches im map-file mit „IMAGEURL" referenziert wird, zu erlauben, z.B:

```
Alias /umn/ "c:/Inetpub/www/umn-www"

<Verzeichnis "c:/Inetpub/www/umn-www">
    Options All
    AllowOverride None
    Order allow,deny
    Allow from all
</Verzeichnis>
```

Internet Information Server

Der IIS wird als Komponente über die Systemsteuerung dem Windowssystem hinzugefügt.

Installation PHP4

Zwei Dateien sind herunterzuladen. Zum einen das Installationsarchiv 'PHP 4.3.4 installer' und das Zip-Archiv 'PHP 4.3.4 zip'. Zuerst wird die Installation mit Hilfe des 'PHP 4.3.4 installer' ausgeführt. Als Installationsverzeichnis ist „c:\PHP" eine gute Wahl. Die Konfiguration des „Internet Information Servers" übernimmt die Installationsroutine. Danach ist das PHP 4.3.4 zip-Archiv zu entpacken und der Inhalt über die Dateien/Verzeichnisse, die sich im Verzeichnis der zuvor installierten php-Version (c:\PHP) befinden, kopiert werden. Einige Dateien sind bereits vorhanden. Diese können einfach überschrieben werden. Anschließend ist der Inhalt des Ordners „../dll", sowie die Datei *php4ts.dll* nach „C:\WINNT\system32" und die *php.ini-dist* in das Verzeichnis „C:\winnt" zu kopieren. Die Datei „php.ini-dist" ist in *php.ini* umzubenennen.

In der Konfigurationsdatei *php.ini* sind folgende Parameter anzugeben:

1. Der Pfad zum Extensions-Verzeichnis (extensions_dir).
2. Folgende Extensionen müssen durch Entfernen des Semikolons am Zeilenanfang eingebunden werden

- extension=php_gd2.dll
- extension=php_domxml.dll

Installation MySQL

MySQL wird für die Client Suite Mapbender benötigt. MySQL ist als Windows Installer auf gewohnte Weise zu installieren.

UMN MapServer 4

Die Firma dmsolutions bietet ein zusammengestelltes Paket mit vorkompilierten dll's und Binaries zum Herunterladen an:

http://www.maptools.org/php_mapscript/index.phtml?page=downloads.html

Entpackt enthält das ZIP-Archiv des UMN-MapServer Version 4.0.1 in etwa folgendes:

Den Ordner „proj“, welcher vollständig nach „c:\“ kopiert werden muss. Die gezippten dlls

xerces_dll.zip
pdfdll.zip
libpq.zip
libcurl-7.10.7_dll.zip
gdal.zip
ECW_DLL.zip

welche entpackt und entweder in das „System32“ der Windows Installation oder in das „cgi-bin“ bzw.„scripts“ Verzeichnis des Webserver parallell zur *mapserv.exe* kopiert werden müssen.

Die *php_mapscript_4.0.1.dll* und *php_proj.dll* müssen in das Extensions-Verzeichnis (C:\PHP\extensions) von PHP kopiert werden.

Zum Schluss muss die *mapserv.exe* in das „cgi-bin“ bzw. „scripts“ Verzeichnis des Webservers kopiert werden.

Apache Standard: „C:\Programme\Apache Group\Apache\cgi-bin“

IIS Standard: „C:\Inetpub\scripts“

Literatur

FreeBSD/OpenBSD

Michael Lucas, Absolute BSD, No Starch Press, August 2002, Englisch
Michael Lucas, Absolute OpenBSD, No Starch Press, Juli 2003, Englisch
Michael Lucas, FreeBSD de Luxe, mitp Verlag, Oktober 2003, Deutsch

Unix/Linux

Arnold Willemer, Wie werde ich UNIX-Guru? - Einführung in UNIX, Linux und Co, Galileo Press, Oktober 2003

Christine Wolfinger, Keine Angst vor UNIX/Linux, Springer, Berlin (April 2002)

* * *

UMN MapServer als OGC-konformer WMS Server

Von Christina Biakowski & Astrid Emde

Zusammenfassung

Der UMN MapServer verfügt über eine Schnittstelle, welche die OGC WMS Spezifikation implementiert.

Im ersten Teil wird kurz beleuchtet, welche Anforderungen die OGC Spezifikation (Theorie siehe Kapitel 3: *OGC und WMS Basis Know How*) konkret an eine Software Architektur stellt und erklärt, warum welche zusätzlichen Komponenten bei der Kompilierung und Installation eingebunden werden müssen. Zusätzlich werden Hinweise für die Fehlersuche gegeben.

Im zweiten Teil wird beschrieben, welche Einträge in der MAP Konfigurationsdatei vorgenommen werden müssen, um den UMN MapServer als OGC WMS Dienst nutzen zu können. Die Konfiguration als WMS Dienst ist die Voraussetzung für eine offene Architektur, die server- und clientseitig mit WMS kompatiblen Diensten aufgebaut werden kann.

Im dritten Teil wird der Einsatz des UMN MapServers als kaskadierender WMS beschrieben, d.h. die Einbindung von Ebenen und Karten „fremder“ WMS kompatibler Kartenserver in MapServer-Karten. Beispielhaft werden die erforderlichen Konfigurations-angaben in der MAP-Datei erläutert.

1 Einführung

WMS (*Web Map Service*) bezeichnet einen über das Internet zugänglichen Dienst oder Service, über den Geodaten in Form von Karten (OGC-G*etMap*-Request) sowie Informationen zu einzelnen Geoobjekten (OGC-G*etFeatureInfo*-Request) angefordert werden können.

Web Map Services ermöglichen es, auf verschiedenen Web Servern vorgehaltene Geodatenquellen im Rahmen einer Cascading WMS Architektur in einer Anwendung zusammenzuführen; somit entfällt die Problematik redundanter Datenbestände. Über das Einrichten verschiedener OGC-WMS kompatibler Map Server können die Daten dezentral von den jeweils zuständigen Stellen vorgehalten werden. Um einen WMS nutzen zu können, braucht man im Grunde nur die zum Ansprechen des Dienstes erforderliche URL zu kennen.

Seit Version 3.5 ist im UMN MapServer die vom OGC (Open GIS Consortium) standardisierte WMS Schnittstelle implementiert. Die aktuelle MapServer-Version (Stand November 2003) unterstützt die WMS-Standards 1.0.0, 1.0.7 und 1.1.0.

Die Kommunikation zwischen Client und WMS-Server erfolgt über standardisierte

Anfragen (Requests). Gemäß OGC-Spezifikation muss ein WMS Server die Requests G*etCapabilities* und G*etMap* beantworten können. Der G*etCapabilities*-Request liefert ein XML-Dokument mit den Eigenschaften des WMS-Projektes zurück. Der G*etMap*-Aufruf liefert eine Karte als Rasterbild zurück. Optional ist die im MapServer implementierte Unterstützung des G*etFeatureInfo*-Requests, über den Informationen über Geoobjekte an einer bestimmten Klickposition angefordert werden können.

Für weitere Informationen zum OGC sowie zur WMS Spezifikation s. Kapitel *OGC und WMS Basis Know How*.

2 Installation von MapServer mit WMS-Support

MapServer kann als Server Web Map Services zur Verfügung stellen sowie als Client innerhalb einer Cascading WMS Architektur Datenquellen anderer WMS Server nutzen.

Ob die verwendete MapServer-Version WMS-Support umfasst, kann durch Abfrage der Versions-Informationen über die Eingabe von *mapserv -v* (Windows) bzw. **.** *Imapserv -v* (Unix) auf der Kommandozeile überprüft werden. Die zurückgegebenen Angaben müssen den Eintrag SUPPORTS=WMS_SERVER bzw. SUPPORTS=WMS_CLIENT enthalten.

Beispiel – MapServer mit WMS-Client und -Server Unterstützung:

```
admin@unix1> ./mapserv -v
```

```
MapServer version 4.0.1 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG
OUTPUT=WBMP SUPPORTS=PROJ SUPPORTS=FREETYPE SUPPORTS=WMS_SERVER
SUPPORTS=WMS_CLIENT SUPPORTS=WFS_SERVER SUPPORTS=WFS_CLIENT
INPUT=EPPL7 INPUT=JPEG INPUT=POSTGIS INPUT=OGR INPUT=GDAL
INPUT=SHAPEFILE
```

Windows

Die zum Download zur Verfügung stehenden Windows-Binaries umfassen in der Regel WMS-Support.

UNIX

MapServer umfasst automatisch WMS-Server-Support, wenn die PROJ-Bibliothek mitkompiliert wird. Für den WMS-Client-Support muss beim Kompilieren *–with-wmsclient* angegeben werden. Wird diese Angabe gemacht, sucht MapServer automatisch nach der ebenfalls erforderlichen *libcurl*-Bibliothek. (s. auch Linkliste).

PROJ epsg-Codes

Nach der OGC-Spezifikation muss ein WMS das Projektionssystem angeben, in dem die bereitgestellten Daten vorliegen, wozu die EPSG-Projektions-Codes verwendet werden.

Um die EPSG-Codes auswerten zu können, benötigt MapServer die *epsg*-Definitionsdatei, in welcher die zu den jeweiligen EPSG-Codes gehörenden Projektionsparameter angegeben sind. Diese Datei *epsg* wird mit den Windows-Binaries bzw. den Sourcen der PROJ.4 Projektionsbibliothek mitgeliefert. Das Default-Verzeichnis für die *epsg*-Definitionsdatei ist unter Windows *c:\proj\nad*, unter UNIX */usr/local/share/proj*.

3 Erstellen eines WMS mit MapServer

Für jeden WMS muss eine eigene MAP-Datei erstellt werden. Dabei handelt es sich um eine ganz "normale" MapServer MAP-Datei, in der bestimmte Parameter und Metadaten-Angaben erforderlich sind. Diese Angaben benötigt MapServer, um auf den *GetCapabilities*-Request ein gültiges XML-Dokument zurückzugeben.

Für das Erstellen eines WMS sind folgende Angaben in der MAP-Datei obligatorisch bzw. empfehlenswert:

Angaben für die gesamte Karte

- Map NAME
- Map PROJECTION
- Map Metadata (innerhalb des WEB Objekts):
    - wms_title
    - wms_onlineresource
    - wms_srs

Im Einzelnen werden die Parameter zur Generierung folgender Angaben im *Capabilities*-Dokument benötigt:

- MAP *NAME* und Metadata *wms_title*

Mit den Parametern werden der *NAME* und *TITLE*-Tag des übergeordneten *root-Layers* im *Capabilities* XML-Dokument befüllt. Das root-Layer im *Capabilities*-Dokument bezieht sich auf die gesamte Karte.

- MAP *PROJECTION* und Metadata *wms_srs*

Ein WMS muss das Projektionssystem angeben, in dem die bereitgestellten Daten vorliegen, wozu die EPSG-Projektions-Codes verwendet werden. Dabei muss “epsg” in Kleinbuchstaben angegeben werden. MapServer nutzt die Angaben aus dem PROJECTION-Block um den (fakultativen) <BoundingBox>-Tag für den übergeordneten *root*-Layer im *Capabilities*-Dokument zu generieren.

- Metadata *wms_onlineresource*

Hier ist die URL anzugeben, über die der WMS auf dem Server zu erreichen ist. Diese

Angabe wird ebenfalls zur Erzeugung des *Capabilities*-Dokuments benötigt.

Beispiel: für WMS erforderliche Angaben in der übergeordneten MAP-Sektion

```
MAP
  NAME "Beispielkarte"
  ...

  PROJECTION
      "init=epsg:31493"
  END
  ...

  WEB
    ...

    METADATA
      "WMS_SRS" "epsg:31493"
      "WMS_ONLINERESOURCE" "http://myserver/umn/mapserv.exe?map=beispiel.map"
      "WMS_TITLE" "Beispielkarte (Kapitel WMS)"
    END
  END
  ...
```

Angaben für jeden LAYER:

- Layer NAME
- Layer PROJECTION
- Layer Metadata
    - wms_title
    - wms_srs

- Layer *NAME* und Metadata *wms_title*

Mit den Parametern werden der *NAME* und *TITLE*-Tag der einzelnen Layer im *Capabilities*-Dokument befüllt. Die Angaben müssen für jede Ebene eindeutig sein. *NAME* wird des weiteren als Parameter im *GetMap* (welcher Layer soll angezeigt werden?) sowie *GetFeatureInfo*-Request (zu welchen Layern sollen Informationen zu den Geoobjekten am Klickpunkt zurückgegeben werden?) übergeben, so dass die Verwendung URL-tauglicher Bezeichnungen empfehlenswert ist (keine Leer- und Sonderzeichen).

- Layer *PROJECTION* und Metadata *wms_srs*

Obwohl nicht obligatorisch, sollten diese Parameter für jede Ebene definiert werden. Generell werden die im übergeordneten MAP-Teil gemachten Angaben zum Projektionssystem vererbt. Liegen die Daten jedoch in einem anderen Projektionssystem vor, sind *wms_srs* und *PROJECTION* unbedingt anzugeben.

Zur erforderlichen Syntax s. o. "Angaben für die gesamte MAP".

Beispiel: für WMS erforderliche Angaben innerhalb der LAYER-Sektion

```
     ...
LAYER
    NAME "Nutzung"
    ...

    PROJECTION
      "init=epsg:4326"
    END
    ...

    METADATA
      "WMS_SRS" "epsg:4326"
      "WMS_TITLE" "Nutzung (Beispieldaten)"
    END

    ...

END
```

Optionale WMS-Parameter

Neben den beschriebenen Pflichtangaben gibt es eine Reihe zusätzlicher optionaler WMS-Metadatenparameter, die in der MapServer-Online-Dokumentation (s. Linkliste *MapServer WMS Server HowTo)* beschrieben sind.

4 Testen des eingerichteten WMS

Nachdem über Hinzufügen der in Kapitel 3 beschriebenen Metadaten in der MAP-Datei ein WMS erstellt wurde, kann dieser durch Eingabe der entsprechenden OGC-WMS Requests (*GetCapabilities, GetMap, GetFeatureInfo*) in einem Webbrowser getestet werden. Die URL für die Requests setzt sich jeweils aus der Online Resource URL der MAP-Datei (angegeben unter Metadata *wms_onlineresource*) sowie den jeweiligen Request-spezifischen Übergabeparametern zusammen.

Das Testen eines WMS kann im folgenden am Beispiel eines vorhandenen MapServer WMS nachvollzogen werden.

4.1 GetCapabilities-Request

Über Eingabe des *GetCapabilities*-Requests im Webbrowser wird die Gültigkeit des erstellten WMS getestet. Der WMS kann angesprochen werden, indem an die Online Resource der MAP-Datei der Parameter *request=GetCapabilities* gehängt wird.

Beispiel:

http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/germany.map&request=getcapabilities

Das zurückgegebene Dokument hat den MIME Typ *application/vnd.ogc.wms_xml*, so dass im Browser in der Regel ein Dialog zum Speichern der Datei erscheint. Nach Speichern der Datei kann diese mit einem Text-Editor oder Browser geöffnet werden und auf ihre Vollständigkeit bzw. Fehlermeldungen (angezeigt mit *WARNING*) überprüft werden.

4.2 GetMap-Request

Aus den Angaben im zurückgegebenen *GetCapabilities*-Dokument kann nun ein *GetMap*-Request erzeugt werden.

Beispiel:

http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/beispiel.map&VERSION=1.1.0&REQUEST=GetMap&LAYERS=Topographie,Grenze&STYLES=default,default&SRS=EPSG:31493&BBOX=2886610.0,5230450.0,4285190.0,6130010.0&WIDTH=600&HEIGHT=400&FORMAT=PNG&BGCOLOR=0xffffff&TRANSPARENT=FALSE

Im Browser wird nun eine Karte mit den angegebenen Layern angezeigt.

Werden an die WMS URL lediglich die Parameter *VERSION=1.1.0&REQUEST=GetMap* gehängt, wird eine Karte mit der in der MAP-Datei angegebenen voreingestellten Größe sowie mit sämtlichen Ebenen mit der Einstellung STATUS ON angezeigt.

4.3 GetFeatureInfo-Request

Im Capabilities-Dokument sind abfragbare Ebenen über das Attribut *queryable = "1"* des *Layer*-Elements gekennzeichnet:

<Layer queryable = "1">

Es gelten die Ebenen als abfragbar, die in der MAP-Datei über eine *TEMPLATE*-Angabe verfügen.

Beispiel für einen GetFeatureInfo-Request:

http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/germany.map&VERSION=1.1.0&REQUEST=feature_info&LAYERS=Postleitzahlbereiche&STYLES=default&SRS=EPSG:31493&BBOX=3419653.6981132077,5647219.727944481,3485624.452830189,5694768.460589551&WIDTH=530&HEIGHT=382&FORMAT=PNG&TRANSPARENT=FALSE&FEATURE_COUNT=30&BGCOLOR=0xffffff&QUERY_LAYERS=Postleitzahlbereiche&X=303&Y=169&INFO_FORMAT=text/html

4.4 Testen im WMS-Client

Ein erstelltes UMN MapServer WMS-Projekt kann in jedem WMS-Client über Angabe der WMS Online Resource bzw. des *GetCapabilities*-Request eingebunden werden.

5 MapServer als WMS Client

Im Rahmen einer Cascading WMS Architektur kann der UMN MapServer Datenquellen von anderen WMS-Servern als Raster-Ebene einbinden. Die verwendete MapServer-Version muss über WMS Client-Support verfügen (s. dazu Kapitel 2: Installation von MapServer mit WMS Support).

Beispiel: Einbindung eines WMS als Layer in die MAP-Datei

```
LAYER
    NAME "WMS_TEST"
    STATUS ON
    TYPE RASTER
    CONNECTIONTYPE WMS
    CONNECTION
"http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/germany.map&version=1.1.0"
    PROJECTION
      "init=epsg:31493"
    END
    METADATA
      "wms_srs" "epsg:31493"
      "wms_title" "WMS_TEST"
     ."wms_server_version" "1.1.0"
      "wms_format" "image/png"
    END
END
```

Die Einbindung eines WMS als Layer in die MAP-Datei erfordert einige WMS-spezifische Metadateneinträge, deren Beschreibung im Einzelnen in der MapServer-Online-Dokumentation (s. Linkliste *MapServer WMS Client HowTo)* nachzulesen ist.

Zusätzlich sind folgende Parameter erforderlich:

- TYPE RASTER
- CONNECTIONTYPE WMS sowie
- CONNECTION.

CONNECTION-String

Als *CONNECTION*-Parameter ist seit Version 4 lediglich die Online Resource URL des einzubindenen WMS anzugeben. Die weiteren WMS Parameter werden aus den Metadata-Angaben generiert. Abweichend davon mussten in den MapServer Versionen 3.5 und 3.6 mit dem *CONNECTION*-String noch weitere WMS-Parameter angegeben werden. (Obwohl obiges Beispiel auf MapServer Version 4 zugreift, musste jedoch zur Anzeige der WMS-Layer *CONNECTION* der WMS Parameter *version* mit übergeben werden).

Über Ergänzen des *LAYERS*-Parameters im *CONNECTION*-String können einzelne Ebenen angefordert werden.

Beispiel:

```
CONNECTION
"http://wms.ccgis.de/umn/bin/mapserv.exe?map=d:/umn/germany.map&version=1.
0.0&Layers=Topographie"
```

Projektionssystem

Fehlerursache bei nicht dargestellten WMS-Layern sind oftmals fehlende bzw. falsche Angaben zur Projektion (PROJECTION-Parameter und Metadata *wms_srs*). Es ist der EPSG-Code des Projektionssystems anzugeben, in dem die WMS-Ebene dargestellt werden soll.

Temporäres Image

Beim Anfordern der WMS-Ebene legt MapServer das vom WMS-Server zurückgegebene Image zunächst lokal als temporäre Datei ab, bevor die Karte gezeichnet wird. Diese temporäre Datei wird im unter IMAGEPATH angegebenen Verzeichnis des Client-Rechners abgelegt und nach Erzeugen der Karte vom MapServer automatisch wieder gelöscht.

6 Link-Liste

Open GIS Consortium und WMS

Open GIS Consortium (OGC) Homepage:
http://www.opengis.org/

Open GIS Consortium (OGC) WMS Implementation Specification:
http://www.opengis.org/docs/01-047r2.pdf

MapServer als WMS-Server und -Client

MapServer WMS Server HowTo:
http://mapserver.gis.umn.edu/doc40/wms-server-howto.html

MapServer WMS Client HowTo:
http://mapserver.gis.umn.edu/doc40/wms-client-howto.html

MapServer UNIX Compilation and Installation HowTo
http://mapserver.gis.umn.edu/doc/unix-install-howto.html#d45e201

MapServer und Projektionen

MapServer Projections & Coordinate Systems:
http://mapserver.gis.umn.edu/doc40/coordinates.html

FAQ Projektionsbibliothek PROJ.4:
http://www.remotesensing.org/proj/faq.html

HTML-Version der MapServer epsg-Datei:
http://mapserver.gis.umn.edu/doc40/proj.html

* * *

WebGIS Client Mapbender – eine D-HTML Lösung

Von Arnulf Christl

Die Hauptarbeit (in Prozessorzeit) innerhalb einer WebGIS Architektur übernehmen die Basiskomponenten auf dem Server oder mehreren Servern. Dazu zählen das Betriebssystem, die Datenbank, der Webserver, die Middleware und zuletzt die Clientapplikation. In diesem Kapitel werden die Clientapplikationen in WebGIS Architekturen beschrieben, die über HTTP im Browser ablaufen.

Mit der Komplexität der gesamten Software der GDI verhält es sich bei dieser Architektur ähnlich: das Betriebssystem ist in seiner Komplexität und Vielfalt unübertroffen, gefolgt von der Datenbank, dem Webserver und Kartenserver. Die am wenigsten komplexe Software im Sinne von Anzahl Zeilen von Programmcode, sind die serverseitigen Skripte, HTML Seiten und JavaScript Code. Sie sind jedoch dafür verantwortlich, dass der Benutzer überhaupt etwas sieht und ermöglichen dem Endanwender den Zugriff auf die Daten. Um diese Skripte, HTML Seiten und optionalen JavaScript Code geht es im folgenden Kapitel.

Statische und dynamisch generierte HTML Seiten

Die Bezeichnung D-HTML fasst eine Softwareschicht zusammen, die mit Hilfe von serverseitigen Skripten dynamisch HTML Seiten erzeugt. In diesem Kontext steht das "D" für "Dynamic" und soll den Unterschied zwischen statisch und dynamisch mit Skripten erstellten HTML Seiten verdeutlichen.

Bei einer statischen Seite wird der HTML Code direkt geschrieben und auf einem Datenträger abgelegt. Auf Anfragen liefert der Webserver den entsprechenden HTML Code an den Browser, der die Seite interpretiert und darstellt.

Dynamische, also D-HTML Seiten werden vollständig oder teilweise von serverseitigen Skripten oder Programmen generiert. Das bedeutet, dass eine Anfrage von einem Browser nicht direkt mit einer fertigen HTML Seite beantwortet werden kann. Es werden dabei in dem Aufruf verpackte Parameter bei der Zusammenstellung der Ergebnis-HTML Seite berücksichtigt und eingebaut. Ein Beispiel für solche dynamischen Komponenten sind natürlich Karten, die nicht statisch als Kacheln irgendwo abgelegt, sondern bei jedem Zugriff neu berechnet werden.

Es gibt verschiedene Skriptsprachen, die HTML Seiten generieren können, zu den bekannteren gehören Perl, PHP und Python.

UMN MapServer MapScript

Der UMN MapServer bietet mit MapScript eine API (Application Programming Interface) zur Erzeugung dynamischer Inhalte und sehr weitgehender Manipulation der Karten. MapScript kann genutzt werden, um eigenständige Applikationen zu erstellen, die eine

sehr hohe Funktionstiefe aufweisen.

MapScript wird für unterschiedliche Skriptsprachen unterstützt, darunter Perl, PHP, Python und auch Java. Die MapScript Technologie wird vor allem für spezielle Anwendungen genutzt, die einen klaren Arbeitsablauf aufweisen und durch die geschlossene Umgebung keine Unterstützung der WMS Spezifikation erfordern.

Auf der Homepage von UMN MapServer gibt es im Bereich <Gallery> eine Auflistung von Applikationen die, mit MapScript in diversen Sprachen implementiert sind.

Applikationen mit WebGIS Clientfunktionalität

Neben dem im folgenden Abschnitt beispielhaft vorgestellten Open Source Projekt Mapbender gibt es eine Vielzahl anderer Client Applikationen, die auf OGC Dienste zugreifen können. Einige dieser WMS Client-Applikationen sind sehr komplex, z.B. können inzwischen auch die meisten Desktop-GIS Produkte WMS Dienste laden.

Schwerpunkt dieses Kapitels sind Clients, die auf dem Open Source Projekt Mapbender basieren und ohne lokale Installation auf dem Client Rechner, direkt im Browser ohne PlugIns ausgeführt werden.

Die Rolle von WMS-Clients in der Gesamtarchitektur

Die Clients sind "Fenster" und Steuerung der Kartenwerke und deswegen von enormer Wichtigkeit für ein Web-basiertes GIS, denn ohne sie würde der Benutzer nichts sehen. Deswegen ist ohne diese letzte, im Falle von D-HTML Clients eher dünne Softwareschicht, die schwerste Datenbank auf dem potentesten Server für den einfachen Anwender nutzlos.

Ein weiterer Grund, warum es Sinn macht, sich mit dieser relativ flachen Schicht in einer GDI-Architektur auseinanderzusetzen ist, dass sie als Schnittstelle zwischen den Diensten (Services), dem Benutzer, aber auch als Konfigurationswerkzeug für den Administrator dient.

Skriptbasierte Applikationen sind auch der Bereich, in dem Programmierlaien am ehesten selbst eingreifen, aber auch die Abläufe in der Gesamtarchitektur erlernen können. Wenn sie als Freie Software oder Open Source lizenziert sind, können sie nicht nur im Quellcode eingesehen, sondern auch verändert werden. Es wird dazu lediglich ein Texteditor benötigt.

Der Client ist das letzte Glied in der Kette einer GDI und bietet damit auch den besten Blick auf das Gesamtgebilde. Ohne die Zusammenhänge der GDI-Dienste in der Architektur zu verstehen, kann diese Software nicht funktionieren. Da die Kommunikation über http erfolgt, bezieht sich diese "GDI-Dienstearchitektur" auf alle GDI-Dienste weltweit.

Die im folgenden beschriebene Software kann also auch als Kommandozentrum für den

weltweiten Zugriff auf Geodaten verstanden werden.

Entwicklungsgeschichte des Mapbender Projekts

Die ersten Vorläufer der Mapbender Architektur gehen auf das Jahr 1998 zurück, wurden zunächst in Perl implementiert und sind aus den Anforderungen der Geoinformationsverarbeitung öffentlicher Verwaltungen entstanden. 2001 wurde die Mapbender Kartenkomponente neu in der Sprache PHP implementiert und wird seit 2003 als Freies Software Projekt betrieben.

In vielen Fällen erschwert die heterogene Architektur der Geoinformationsverarbeitung in öffentlichen Verwaltungen den Aufbau eines einzigen monolithischen Auskunftssystems im Intra- und Internet. In der Regel haben verschiedene Ämter zunächst weitgehend unabhängig voneinander raumbezogene Informationen erfasst. Dabei werden Produkte unterschiedlicher Hersteller eingesetzt, was dazu führt, dass die Daten in verschiedenen, meist untereinander inkompatiblen Formaten vorliegen.

Diese Karten und Sachdaten werden unter Umständen bereits mit proprietären Kartenservern des gleichen Softwareherstellers als Webdienst zur Verfügung gestellt. Allerdings besteht oft der Wunsch, die Kartenwerke auch an Arbeitsplätzen anderer Ämter der Verwaltung bereitszustellen, ohne die gleiche proprietäre Software erwerben zu müssen.

In einem weiteren Schritt sollen einige Daten auch Externen zur Verfügung gestellt werden, um sie direkter in den Verwaltungsprozess einzubinden und den Zugriff auf Daten zu vereinfachen. Diese Möglichkeit soll allerdings sehr differenziert zur Verfügung stehen. Die Berechtigung, auf welche Karten zugegriffen werden kann, welche Datenabfragen zur Verfügung stehen und welche Daten verändert werden dürfen, hängt von der jeweiligen Nutzergruppe oder von den individuellen Berechtigungen der einzelnen Mitarbeiter ab.

Zusätzlich sollen verstärkt auch dem Bürger Geoinformationen über Internettechnologie zur Verfügung gestellt werden. Die so entstehenden Auskunftssysteme sollen Bestandteil des Internetangebotes werden und möglichst nahtlos mit den anderen Informationen der städtischen Webseite vernetzt werden können.

Die aus diesen heterogenen Anforderungen gewachsene komplexe Architektur erfordert eine zentrale Verwaltung, welche die Zugriffsrechte regelt, und die zentrale Administration aller Datenquellen ermöglicht.

Die konsequente Unterstützung der OGC WMS Spezifikation kann das gewährleisten. Aus diesem Grund unterstützt die Mapbender Software nicht nur die lupenreine OGC WMS Spezifikation, sondern auch Implementierungen, die kleinere Fehler aufweisen. Dadurch kann bereits ein sehr breites Spektrum an unterschiedlichen Softaresystemen unterstützt werden. Der Anpassungsaufwand, um leicht inkonsistente WMS Dienste zu berücksichtigen, ist in Mapbender relativ gering. Zusätzlich werden im Kundenauftrag ständig weitere Anpassungen dafür vorgenommen.

Systembeschreibung der Mapbender Software

Die Mapbender Software ist eine modulare, webbasierte, plattformunabhängige, serverseitig in PHP implementierte WebGIS Client Suite. Mapbender stellt Module zur Verwaltung von "Karten" (OGC WMS-Dienste) zur Verfügung und enthält eine Benutzer- und Gruppenverwaltung. Diese "Karten" oder "Kartendienste" können verwaltet, konfiguriert und miteinander überlagert oder verschnitten werden. Die Mapbender Software wird vollständig über Webseiten bedient und verwaltet.

Lizenzbedingungen und Links zu anderen Projekten

Das Projekt Mapbender ist nach der GNU General Public License (GPL) lizenziert und wird auf der freien Plattform SourceForge bereitgestellt:

(http://sourceforge.net/projects/mapbender/).

Serverseitig ist die Mapbender Client Suite mit der Skriptsprache PHP implementiert:

(http://www.php.net/)

Für die Datenhaltung der Benutzer- und Projektverwaltung wird eine relationale Datenbank benötigt, das aktuelle Mapbender Release (01/2004) enthält eine ODBC Implementierung (z. B. für MS Access) und unterstützt auch die Open Source Datenbank MySQL:

(http://www.mysql.de/)

Um die PHP Skripte in einem Browser anzeigen zu können, wird ein CGI-fähiger Webserver benötigt, z.B. das Apache http Server Project (ab Version 1.3.x) oder der Microsoft IIS.

OGC Kompatibilität

Das aktuelle Release (01/2004) unterstützt die WMS Spezifikationen 1.0.0, 1.1.0 und 1.1.1 .

(http://www.opengis.org/)

Bitte beachten Sie, dass die Mapbender Client Suite nicht OGC WMS kompatibel im Sinne einer Zertifizierung ist. Um diese zu erwerben ist es erforderlich, die zu zertifizierende Software mit einer Prüfsoftware zu testen, die mit einer standardisierten

Teststellung die Kompatibilität prüft. Die Lizenz dieser Prüfsoftware ist aber proprietär, Zitat aus der Lizenz:

„ ...2.5 That the Licensee acknowledges that the Test Engine is proprietary and confidential to the Open Group...“Für diese Lizenz werden Kosten erhoben. Es lohnt sich an dieser Stelle, folgende Rechenbeispiele anzuschauen:

http://www.opengis.org/resources/?page=testing&view=testfees

Hier ergibt sich die leicht absurde Situation, dass eine Open Source Software für eine Open GIS Zertifizierung eine proprietäre Software lizenzieren muss.

Bei der Implementierung von Mapbender wurde eine pragmatischere Lösung gefunden. Die Mapbender Client Suite "versteht" mehrere "WMS Dialekte", wie sie von gängigen (proprietären und freien) Kartenserver-Produkten implementiert werden.

Der Projektname

Der Name des Projekts Mapbender setzt sich aus dem Wort "Map" (englisch für "Karte") und dem Verb "bend" (englisch für "biegen") zusammen. Der "Kartenbieger" ist in der Lage, alle Kartenwerke so zu verbiegen, dass sie zusammenpassen. Geometrische Verzerrungen sollen allerdings nicht durch sinnlose Gewalteinwirkung, sondern unter Berücksichtigung der entsprechenden Projektionssysteme erfolgen.

Die subkutanere Bedeutung ist in Anlehnung an die Zeichentrickserie Futurama zu sehen. Einer der Hauptprotagonisten, ein alkoholkranker, kleptomaner Stahlträger-Biegeroboter sorgt durch Eigenwilligkeit und Nonkonformismus für allerlei Spaß, sein Name ist Bender.

Der Funktionsumfang des Mapbender Projekts

Der Funktionsumfang des Mapbender Projektes teilt sich in drei Bereiche. Der Bereich "Geo-Administration" dient der Erstellung, Verwaltung und Pflege von Kartendiensten, sowie der Verwaltung von Benutzern und Gruppen. Diese Komponenten der Mapbender Software weden von Geo-Administratoren genutzt, um kartenbasierte Anwendungen zu erstellen, Kartendienste zu verwalten, Benutzer anzulegen und ihnen Zugriff auf Geodaten zu erteilen. Der Geo-Administrator kann Mapbender ähnlich wie ein Desktop GIS nutzen, Daten-Dienste laden und in engen Grenzen auch aufbereiten.

Die Benutzeroberflächen mit Kartenfunktionalität, also die eigentlichen GIS-Clients, werden den Benutzern von der Geo-Administration bereitgestellt. Die Endanwender arbeiten mit diesen Anwendungen ähnlich wie mit einem herkömmlichen Desktop GIS, genießen allerdings einen wesentlich höheren Komfort, da sie sich nicht mit der Aufbereitung der Geodaten oder der Implementierung von speziellen Funktionen aufhalten müssen.

Der dritte Bereich umfasst Software aus den Bereichen Datenerhebung, Organisation, Metainformation und enthält spezielle Softwaremodule für die Erweiterung und Einrichtung

von Schnittstellen zu lokalen Applikationen, die noch nicht über Netztechnologie verfügen.

Im Folgenden werden diese Bereiche des Mapbender Projekts vorgestellt und einzelne Funktionen näher erläutert.

Geo-Administration

Die Administration teilt sich in mehrere Bereiche, dazu zählen die Benutzerverwaltung, WMS- und Projektverwaltung und Berechtigungen.

Benutzerverwaltung

Nach der Installation enthält die Benutzerverwaltung zunächst einen einzigen User. Mit dieser Standard-Anmeldung können neue Benutzer hinzugefügt, gelöscht, geändert und Gruppen zugeordnet werden. Gruppen und Benutzern können spezielle Rechte zugeordnet werden, z.B. ein bestimmtes Projekt sehen, Daten abfragen oder auch Projekte administrieren. Der Benutzer erbt alle Rechte der Gruppe, aber nicht umgekehrt. Ein Benutzer sieht also immer alle Gruppenprojekte plus die eigenen. Jeder Benutzer kann Mitglied mehrerer Gruppen sein.

WMS- und Projektverwaltung

Mit der Projektverwaltung können Dienste angelegt, verändert und gelöscht werden. Das "Anlegen" eines Projektes bedeutet, dass ein WMS "hochgeladen" wird. Dabei wird ein OGC WMS Capabilities-Dokument angefordert, analysiert und in die Administrations-datenbank eingetragen. Im aktuellen Release (01/2004) werden die WMS Spezifikationen 1.0.0, 1.1.0 und 1.1.1 unterstützt (Erläuterung siehe OGC Kompatibilität). Zusätzlich zu den WMS konformen Diensten werden auch einige lediglich WMS kompatible Dienste unterstützt.

Um die Karten eines WMS Dienstes Benutzern über die graphische Benutzeroberfläche von Mapbender bereitzustellen, wird zunächst ein neues Projekt angelegt, in das die WMS Dienste eingebunden werden. Durch den Prozess des Hochladens können auch nicht-WMS konforme Dienste unterstützt werden, da sie von Mapbender zunächst in dem eigenen erweiterten Datenmodell abgebildet werden. Hier können auch manuelle Änderungen an den Diensteinstellungen vorgenommen werden.

Dem so entsandenen Projekt können Benutzer und Gruppen zugeordnet werden, die dadurch das Recht erhalten, die enthaltenen Karten und Daten anzuzeigen.

Dieses Vorgehen wurde gewählt, um die "nackten" WMS Dienste möglichst effektiv vor dem Endanwender zu kapseln, da sie oft hässlich, schlecht zu strukturieren und sperrig zu handhaben sind.

Dem Geo-Administrator kann der Umgang mit diesen Diensten aber ohne weiteres zugemutet werden, es unterscheidet sich letztendlich nicht davon, eine Geodaten-Datei

aus einem Ordner in ein Programm zu laden oder die eigenen Internet Bookmarks zu verwalten.

In den Projekten können die Dienste mit unterschiedlichen Grundeinstellungen eingebunden werden. Ein physikalischer WMS Dienst kann damit für unterschiedliche Nutzer mit differenzierter Funktionalität bereitgestellt werden.

In einem Geobasisdaten Projekt könnte z.B. die Berechtigung zur Anzeige der Flurkarte (oder ALK) allen Nutzern der Stadtverwaltung zugeordnet werden, der Zugriff auf die Eigentumsnachweise (ALB) jedoch nur bestimmten Benutzern oder einer speziellen Gruppe gestattet sein.

Projekteigenschaften

Einige Eigenschaften eines WMS Dienstes können angepasst und eingestellt werden. Dabei stehen selbstverständlich nur die im Capabilities-Dokument angebotenen Formate des jeweiligen Dienstes zur Verfügung:

- Bildformat (in der Regel PNG, GIF oder JPEG)
- Koordinatenbezugssysteme (Spatial Reference System); als Standard wird (fast) immer WGS84 (EPSG:4326) angeboten, zusätzlich können auch projizierte Bezugssysteme ausgewählt werden, z.B. Gauß-Krüger Streifen 3 (EPSG:31493 oder neuerdings EPSG:31467).
- Dienstebenen in einem WMS können das Attribut <queryable = 1> setzen, dann antwortet der WMS Dienst auf einen FeatureInfo-Request. Diese Eigenschaft kann in Mapbender explizit unterbunden werden.
- Es kann eingestellt werden, welche Ebenen beim Start des Projektes bereits aktiviert sind, d.h. gezeichnet werden.
- Es kann eingestellt werden, ob dem Benutzer gestattet wird Ebenen zu aktivieren oder zu deaktivieren (an- bzw. ausstellen).
- Die Reihenfolge, in der die Ebenen geschichtet werden (übereinander liegen), kann individuell angepasst werden.

Zusätzlich können weitere Attribute, die nicht in der WMS Spezifikation abgebildet sind, hinzugefügt werden. Dazu gehören:

- die Möglichkeit, ein Copyright in die Karte zu integrieren.
- eine globale Grenze mit einem minimalen und maximalen Maßstab, der beim Browsen nicht unter- oder überschritten werden kann. Damit wird verhindert, dass sich Anwender „in die Umlaufbahn schießen“ oder mikroskopisch kleine Ausschnitte anfordern.
- Eine Maßstabsleiste.

Berechtigungen

Alle Anwendungen werden über Berechtigungen gesteuert. Um einem Anwender oder einer Gruppe z.B. einen hochauflösenden Druck anzubieten, muss diese Funktionalität zunächst als PHP Modul in die Skriptbibliothek eingebunden werden. Wenn ein Benutzer mit den entsprechenden Rechten die graphische Benutzeroberfläche öffnet, wird dynamisch der neue Knopf mit der hochauflösenden Druckfunktionalität eingebunden.

Diese Logik wird in Mapbender beständig weiter ausgebaut und soll dem Anwender nach und nach eine immer individueller konfigurierte Arbeitsumgebung bieten.

Kartenoberfläche

Neben der Verwaltungsschicht bietet Mapbender dynamisch konfigurierte Oberflächen mit den gängigen Navigationswerkzeugen und optional zusätzlich hinzuschaltbaren Werkzeugen. Die Navigation beinhaltet beispielsweise das Zoomen auf eine Klickposition in der Karte, das Aufziehen einer Zoombox, das Verschieben der Karte, einen Navigationsrahmen und optional eine Maßstabsauswahl und Maßstabsbalken. Hinzu kommen optionale Module zur Streckenmessung und zum Druck in verschiedenen Qualitätsstufen.

Das Kartenfenster ist in einigen graphischen Benutzeroberflächen dynamisch einstellbar und ermöglicht die individuelle Anpassung der Kartengröße durch den Benutzer. Zusätzlich kann Benutzern die Möglichkeit eingeräumt werden, andere Kartenwerke hinzuzuladen oder den aktuellen Zustand eines Arbeitsprojekts zu sichern.

Das Mapbender Projekt enthält mehrere Oberflächen,angefangen bei einer HTML 3.2 kompatiblen Version, die ohne JavaScript auskommt, bis hin zum Client mit eingebundenem Java Applet.

PDA Client – Basis-Oberfläche ohne JavaScript

Manche Browser verstehen kein JavaScript, z.B. laufen auf älteren PDA (Personal Digital Assistant) Browser, die lediglich HTML 3.2 verstehen. Außerdem gibt es Meinungen, die vertreten, dass JavaScript ein Sicherheitsrisiko darstellt. Des weiteren kann der Verzicht auf JavaScript bei gleichzeitig gezielter Verschlankung der HTML Seiten die Menge an Netzverkehr enorm verringern. Das ist vor allem bei langsamen Netzleitungen wichtig, besonders bei mobilem Einsatz über GSM (**G**lobal **S**ystem for **M**obile Communications) oder dem paketorientierten und etwas schnelleren GPRS (**G**eneral **P**acket **R**adio **S**ervice) Dienst.

Der PDA-Client wird automatisch gestartet, wenn aus Sicherheitsüberlegungen die Ausführung von JavaScript im Browser deaktiviert ist. In der Anmeldung ist eine Weiche, die nicht-JavaScript-fähige Browser umleitet.

Kartenfenster und Navigation

Das Kartenfenster ist für eine Auflösung von 240 x 320 Pixel voreingestellt und enthält einen schmalen, navigierbaren Rahmen. Die Auflösung kann für unterschiedliche Ausgabegeräte angepasst werden, z.B. auf 208x320 für Smartphones der neuen Generation.

Die Oberfläche wurde bewusst von optischen 'Gimmicks' freigehalten, was einen schlanken HTML Code ergibt und den schmalen Bandbreiten von mobilen Verbindungen Rechnung trägt. Einer optischen Aufpeppung steht prinzipiell nichts im Weg, allerdings werden diese mobilen Systeme normalerweise im professionellen Umfeld genutzt. Mit der Einführung von UMTS sollte sich diese Technologie eigentlich auch für den Enduser durchsetzen, was den Bedarf an "hübschen" Anwendung erhöhen würde.

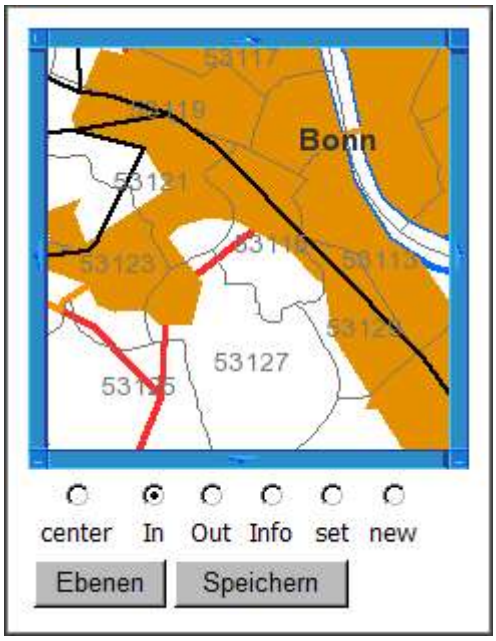

Abbildung: Mapbender PDA Client Oberfläche

Die Kartennavigation erfolgt über eine Vorauswahl der Funktionen: Center, Zoom-In oder -Out, die alle einen WMS getMap-Request absetzen. Die Knöpfe <set> und <new> sind Bestandteil einer PDA Oberfläche zur Abfrage, Erhebung und Pflege von Baumstandorten mit einem Pocket-PC mit GPS Anschluss.

Ebenenübersicht und Auswahl

Über den Knopf <Ebenen> wird die Ebenenübersicht angezeigt. Hier können die Ebenen einzeln an- und ausgeschaltet werden oder zur Abfrage markiert werden (s. Abbildung Ebenenauswahl).

Abbildung Ebenenauswahl: Mapbender PDA Client Ebenenauswahl

Beim Klick auf den Knopf <Karte> wird wieder das Kartenfenster geladen und gleich mit den neuen Einstellungen gezeichnet.

Sachdatenabfrage (getFeatureInfo-Request)

Über den Info-Knopf in der Kartenoberfläche kann die Sachdatenabfrage ausgewählt werden. Der nächste Klick in die Karte setzt einen WMS getFeatureInfo-Request ab. Die Ergebnisse dieser Anfrage werden im gleichen Fenster unter der Karte ausgegeben, so dass gleich die nächste Aktion durchgeführt werden kann. Alternativ kann hier auch eine Folgeverarbeitung ansetzen, in der z.B. eine Baumkataster-Datenmaske angezeigt wird, in die Eigenschaften der Bäume während der Begehung online eingetragen werden können.

Standard Oberfläche mit JavaScript

Die Standard Oberfläche ist die am weitesten verbreitete Lösung aus dem Mapbender Projekt. Die Oberfläche orientiert sich optisch an den Bedürfnissen herkömmlicher Desktop GIS Anwender. Folgende Steuerelemente sind Bestandteil der Oberfläche:

- Ebenenübersicht
- Kartenfenster und Übersichtskarte
- Werkzeugleiste

- Statuszeile

Ebenenübersicht

Die Ebenenübersicht oder auch GDE (Geo Data Explorer) listet alle verfügbaren Ebenen auf (s. Abbildung GDE). Wenn Ebenen aus mehreren WMS eingebunden werden, wird der Titel des entsprechenden WMS Dienstes als Überschrift über die Ebenennamen gesetzt, um die GDE zu strukturieren (in dem abgebildeten Beispiel ist <Germany> der Titel des WMS).

Abbildung GDE: Mapbender Ebenenauswahl (GDE)

Die linke Spalte zeigt an, ob diese Ebene in der Karte dargestellt werden soll oder nicht. Die rechte Spalte zeigt an, dass diese Ebene bei einer Sachdatenabfrage (getFeatureInfo-Request) berücksichtigt werden soll.

Die Ebenennamen sind als Hyperlink ausgeprägt, der voreingestellt eine Legendenansicht zu dieser Ebene anzeigt. Alternativ können hinter dieser URL auch Metadaten nach ISO 19115 oder weiterführende Links eingebunden werden.

Kartenfenster und Übersichtskarte

Das Kartenfenster enthält ein oder mehrere überlagerte Rasterbilder der abgebildeten WMS-Dienste und wird beim Neuzeichnen aktualisiert. Um das Kartenfenster herum liegt ein navigierbarer Rahmen, der bei einem Klick die aktuelle Position um 50% in die jeweilige Richtung verschiebt (s. Abbildung Kartenfenster).

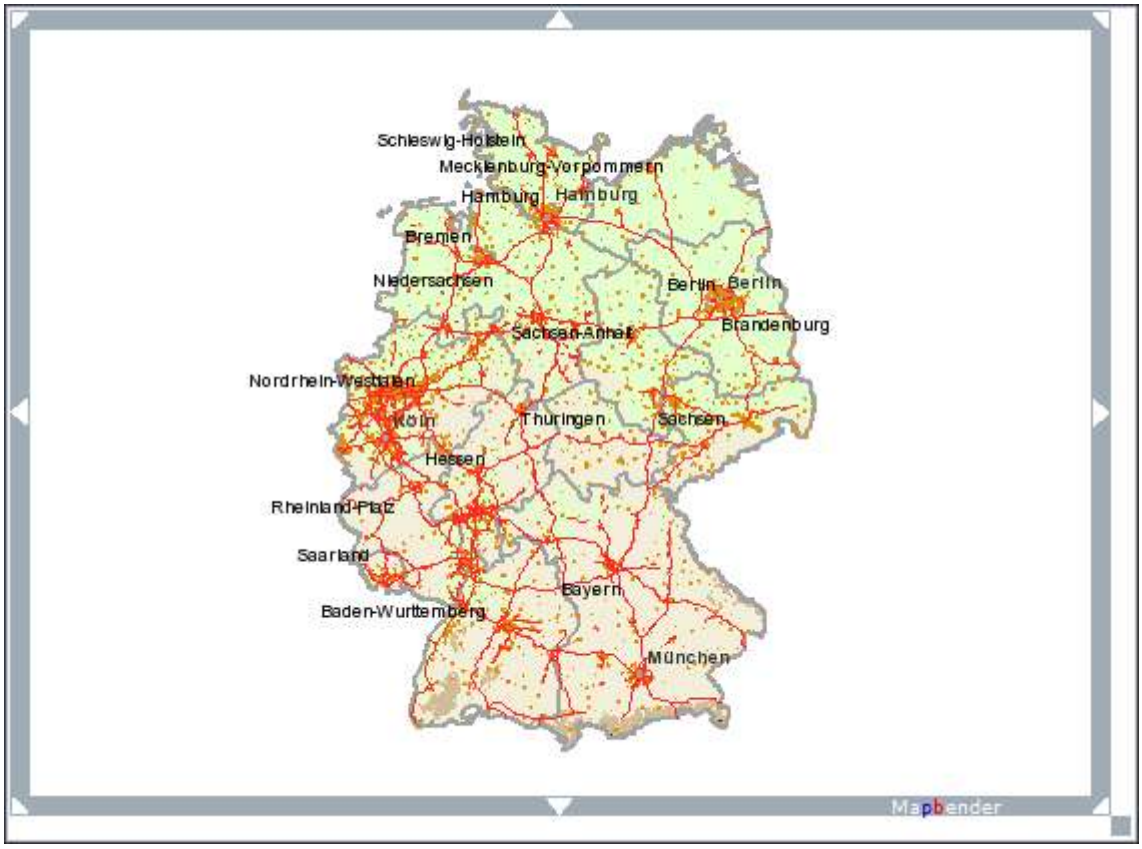

Abbildung Kartenfenster: Mapbender Kartenfenster

Das Kartenfenster kann diverse zusätzliche Aufgaben übernehmen, z.B. Markierungselemente anzeigen, im Client digitalisierte Objekte darstellen, Messpunkte und Strecken abbilden, etc.. Duch die Nutzung von i-Frames können Ergebnisse einer Sachdatenabfrage auch direkt in das Kartenfenster eingeblendet werden.

Über den Knopf in der rechten, unteren Ecke kann die Größe der angeforderten Kartenbilder per Drag&Drop angepasst werden.

Werkzeugleiste

Die Werkzeugleiste wird dynamisch erstellt und enthält alle Knöpfe und Funktionen, die sich aus den Berechtigungen des angemeldeten Benutzers ableiten. In dem abgebildeten Beispiel sind lediglich die üblichen Standardfunktionen eingebunden, die sich auch in jedem Desktop GIS wiederfinden.

Abbildung Werkzeugleiste: Mapbender Werkzeugleiste (Standard)

Die Funktionen von links nach rechts: Neuzeichnen, letzte Ansicht, Zoom-In, Zoomfaktor, Zoom-Out, Zentrieren, Verschieben, Abfragen, Suchen, Abstand messen, Linie messen, Drucken, Online Hilfe, Maßstabsanzeige und Auswahl, Maßstabseingabefeld und Abmelden.

Statuszeile

In der Statuszeile des Browsers, oder in einer eigenen Textzeile können Informationen eingeblendet werden. Hier kann z.B. die aktuelle Mausposition in Weltkoordinaten angezeigt werden, Abfrageergebnisse eines getFeatureInfo-Requests oder Hinweise zur Bedienung der Oberfläche eingeblendet werden.

Individuell angepasste Oberflächen

Die Offenheit der Mapbender PHP Implementierung ermöglicht es, individuelle Oberflächen für spezielle Anwendungen zu erstellen. Dabei können eigene Steuerelemente entwickelt, in die HTML Oberflächen eingebunden oder bestehende Objekte genutzt werden. Der Vielfalt an Oberflächen sind praktisch keine Grenzen gesetzt.

Mapbender SPA (ein Stadtplandienst)

Die Mapbender SPA Oberfläche (s. Abbildung SPA) wurde im Rahmen des MEDIA@Komm Projektes realisiert. Die Oberfläche ist für die Benutzung als Stadtplandienst durch Nutzer aus dem Internet optimiert.

Die Hintergrundkarten werden von unterschiedlichen Diensten bereitgestellt und nahtlos mit den Dienstebenen, die von einem UMN MapServer aufbereitet werden, integriert.

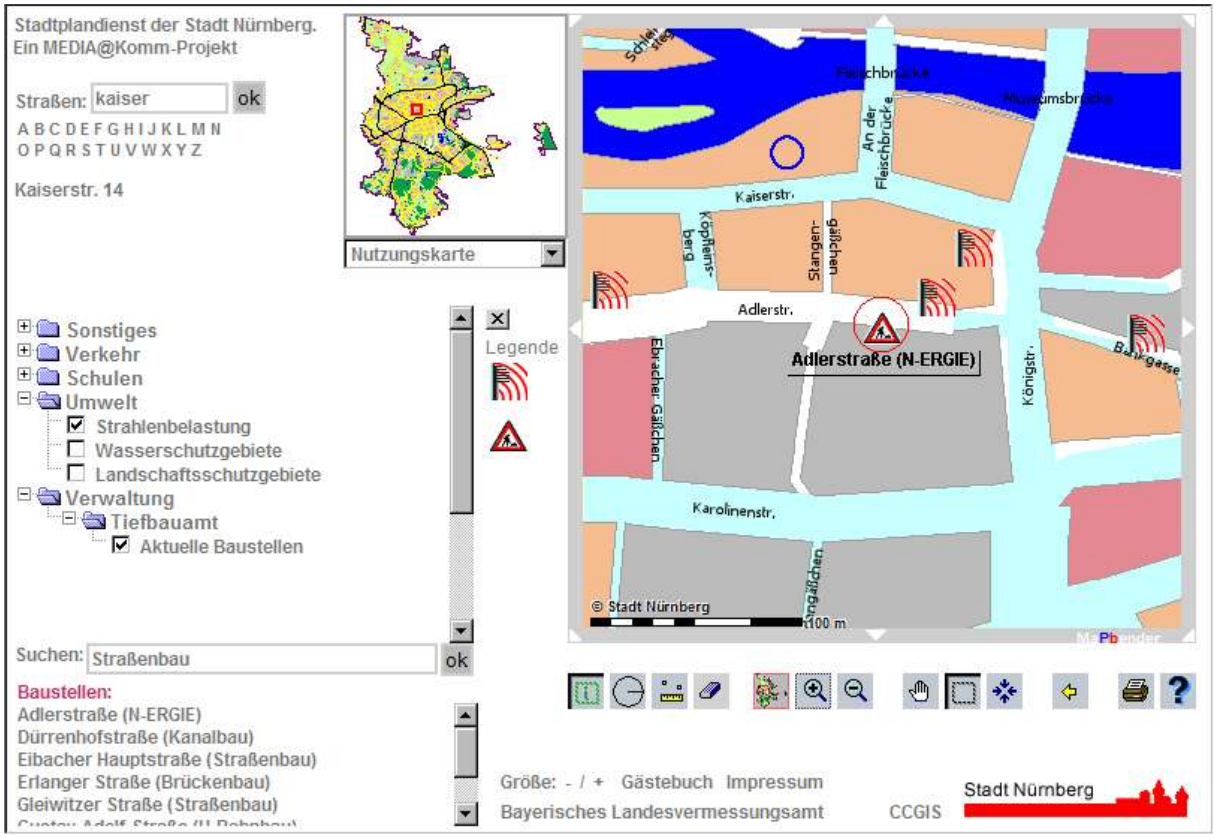

Abbildung SPA: Mapbender SPA Oberfläche

Straßen- und Objektsuche

Die Straßensuche ist ebenso direkt in die Oberfläche integriert wie die Suche nach Objekten. Beide Suchbereiche bieten die gleichen Funktionen, wurden aber bewusst in zwei getrennte Bereiche aufgeteilt, um dem Anwender die Möglichkeit zu geben, Suchergebnisse gemeinsam zu visualisieren. In einem weiteren Schritt wäre es naheliegend, eine Navigation von einem gefundenen Objekt zu einem anderen zu integrieren.

Funktionale und kartographische Oberflächenelemente

Praktisch jedes Objekt im Stadtplandienst ist mit einem Link versehen, der weitere Informationen anzeigen kann. Ein Klick auf die Ebenennamen (oder Themen) in der GDE öffnet ein neues Fenster, in dem Metadaten angezeigt werden. Ein Klick auf ein Objekt in der Karte öffnet ein neues Fenster mit dem Abfrageergebnis und Links zu weiteren Informationen. Die künstliche Trennung zwischen Anwendungsoberfläche und

Karteninhalt verliert zunehmend an Bedeutung, da beide relevante Informationen des Stadtplandienstes enthalten.

Werkzeugleiste und Status- oder Infozeile

Die Werkzeugleiste ist unter das Kartenfenster verlegt worden. Das widerspricht zwar der Gewohnheit vieler GIS Anwender, wird jedoch von GIS-unerfahrenen Benutzern bestens angenommen, da die Werkzeugleiste sowieso eher selten benutzt wird. Der normale Einstieg in die Stadtplananwendung erfolgt über eine Adresse oder durch einen direkten Link von der Homepage der Stadtverwaltung.

Abbildung Werkzeugleiste: Mapbender SPA Werkzeugleiste und Statuszeile

Die Status- und Infozeile wurde direkt über die Werkzeugleiste verlegt, um dem Anwender Informationen zu den einzelnen Funktionen zu erläutern ("Kartenausschnitt zeigt mehr Details...") während der Mauscursor (in dem Beispiel ein Hand-Symbol) darüber schwebt. Dadurch erlernt der Anwender während der Benutzung die Bedienung des Systems, statt sich vorher oder in einem separaten Fenster darüber informieren zu müssen.

Kartenfenster als eingebundenes Java Applet

Bei einem Monitoring-System (siehe Abbildung unten) wurde in Zusammenarbeit der Firmen CCGIS und DIALOGIS GmbH das herkömmliche Kartenfenster durch ein Java-Applet ersetzt, um den Benutzer erweiterte Maus-Funktionalität anzubieten und dynamisches Highlighting zu implementieren.

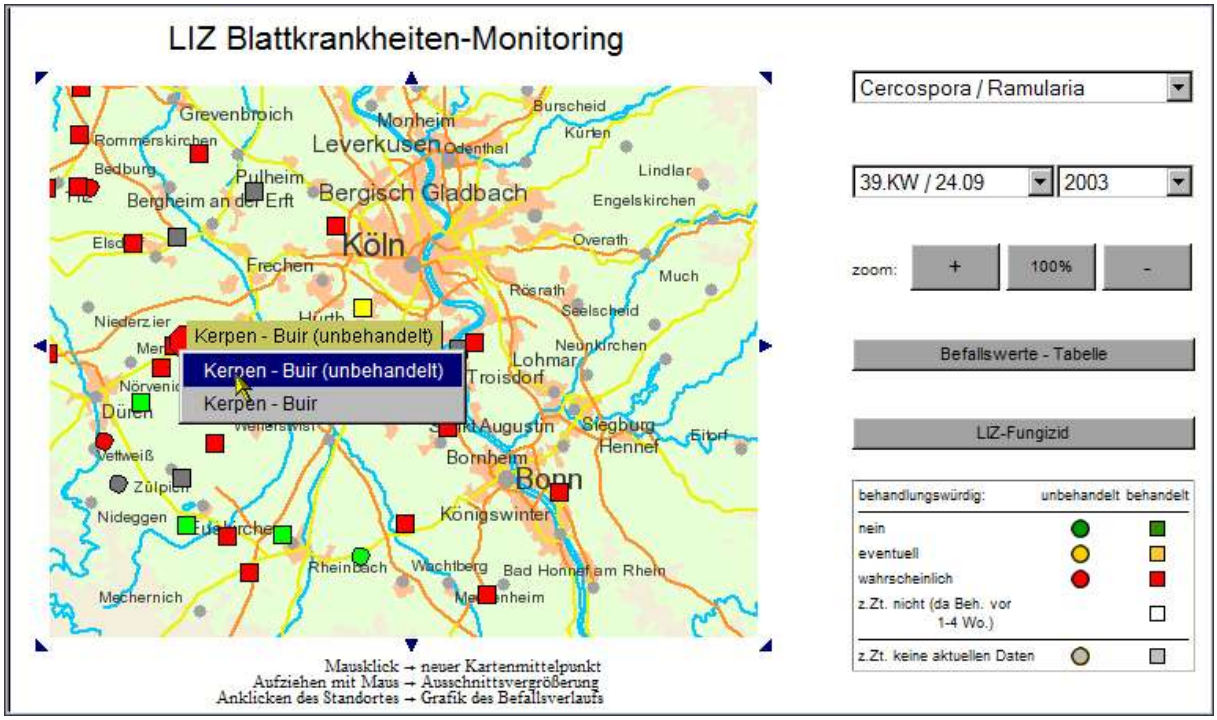

Abbildung Monitoring: Mapbender Monitoring System mit eingebundenem Java Applet

In dem abgebildeten Beispiel wird dem Nutzer ein Kontextmenü angeboten, aus dem verschiedene Objekte gewählt werden können. Die Anwendung wird als verteiltes System zur Eingabe, Pflege und Überwachung von Blattkrankheiten bei Zuckerrübenpflanzen betrieben. Die Boniteure prüfen die einzelnen Ackerschläge im Wochenrhythmus und können alle erhobenen Daten von jedem Internet-Arbeitsplatz aus eingeben. In der Zentrale werden die Eingaben überwacht. Der Landwirt kann über Internet die Angaben jederzeit einsehen, um sich zu informieren, sowie prüfen und gegebenenfalls Meldung an die Zentrale geben, falls Werte fehlen oder falsch eingegeben wurden.

Die Produktivität des gesamten Prozesses konnte durch die Einführung des Systems erheblich beschleunigt werden, gleichzeitig wird durch die offene Herangehensweise ein wesentlich höherer Qualitätsstandard des Dienstes erzielt.

Spezialanwendungen

Es wurden bereits diverse Spezialanwendungen mit der Mapbender Architektur implementiert. Dabei können unterschiedlichste Technologien von eingebundenen Java Applets bis hin zu Active-X Komponenten zum Einsatz kommen. Die Komponenten werden in die bestehende Infrastruktur aus PHP Skripten eingebunden, die über das Mapbender Datenmodell auf alle relevanten Daten der referenzierten WMS Dienste zugreifen.

Kommunikation mit lokalen Anwendungen über Active-X Technologie

Auch wenn Active-X als unsichere Technologie gilt und im Normalfall nur in reinen Microsoft Windows Umgebungen genutzt wird, bietet der direkte Zugriff auf COM Objekte eine hohe Integration von Daten und Methoden lokaler Applikationen mit Webanwendungen. Die CCS_Connector Software (ein Unterprojekt von Mapbender) enthält alle für eine Active-X Anbindung erforderlichen Codebestandteile inkl. der erforderlichen Kommunikation.

Jede Windows-Applikation verfügt über eine eindeutige Bezeichnung in der Windows-Registry (CLSID). Wenn der Internet Explorer diese Class-ID in einem speziellen Tag der HTML Seite findet, wird automatisch eine API Verbindung aufgebaut. Über diese kann die Webanwendung direkt mit der lokalen Anwendung kommunizieren.

Dadurch wird es ermöglicht, dass aus der Webanwendung heraus lokale Applikationen gesteuert werden können. Wenn in der Webanwendung die Objekt-ID eines Geo-Objektes gefunden wurde, kann diese an die lokale Applikation übermittelt werden und eine Folgeverarbeitung anstoßen. Umgekehrt kann die lokale Applikation die Webanwendung ebenfalls ansprechen und z.B. bestimmte Ebenen aktivieren und einen Koordinatenausschnitt anfordern.

Über diese Technologie können bestehende Arbeitsabläufe beibehalten und um einen GIS Anschluß erweitert werden, ohne wieder auf proprietäre Schnittstellen setzen zu müssen.

Das ist vor allem für Hersteller von Applikationen mit einem hohen Verbreitungsgrad interessant, weil die Entwicklung von Schnittstellen für diverse GIS entfällt und eine einfache Kommunikation über die Mapbender Architektur mit beliebigen WMS Diensten erfolgen kann.

Die Weiterentwicklung von Mapbender

Seit der Befreiung der Mapbender Software als GNU GPL-lizenziertes Projekt hat die Entwicklung starke Impulse bekommen. Neben vielen Spezialanwendungen hat davon auch der Standard Client profitiert. Die nächsten bereits angefangenen und teilweise beauftragten Entwicklungsstufen sollen im Sommer 2004 abgeschlossen und in die produktive Version übernommen werden.

Code-Modularisierung

Von Anfang an sollte der Code möglichst modular und wiederverwendbar sein. Bereits das zweite Reengineering hat einige Schwachstellen beseitigt und die Einbindung von Modulen erleichtert.

Über eingebundene PHP Module kann der Funktionsumfang von Mapbender Clients

bereits in der aktuellen Version erweitert werden, allerdings ist es immer noch erforderlich in den JavaScript Dateien Einträge vorzunehmen und je nach Anforderung aus- oder einzukommentieren.

Objektorientierter Ansatz

Das dritte Reengineering im ersten Halbjahr 2004 stellt alle Funktionen und Eigenschaften über Klassen und Methoden bereit. Der initial höhere Entwicklungsaufwand und auch komplexere Code wird durch erheblich einfachere Pflegbarkeit ausgeglichen. Mit inzwischen mehreren hundert produktiven Installationen weltweit ist der Bedarf nach einfacher Updatefähigkeit stark angestiegen. Viele Anwendungen haben sich inzwischen parallel weiterentwickelt und sollen in dem neuen Mapbender Softwaredesign wieder zusammengeführt werden.

Erweiterung des Projektkonzepts

Die bisherige Kapselung von WMS-Diensten in Projekten wird konsequent erweitert. Bisher war es möglich, einen WMS Dienst direkt einem Benutzer zuzuordnen. Das wird in der nächsten Version nicht mehr möglich sein, statt dessen können Benutzer nur noch Projekten inkl. Benutzeroberfläche zugeordnet werden. Ein Projekt mit Benutzeroberfläche, intern kurz als GUI (Graphic User Interface) bezeichnet, enthält immer mindestens einen WMS Dienst und eine Kartenoberfläche, die über die Geo-Administration zusammengestellt werden.

Damit wird eine wesentlich höhere Flexibilität bei der Zusammenstellung von Anwendungen bei gleichzeitig höherer Datenkonsistenz der Benutzer und WMS Eigenschaften erzielt.

Erweiterung des Mapbender Datenmodells

Das Mapbender Datenmodell für die Oberflächengestaltung wird umfangreich erweitert. Alle Oberflächenelemente werden ausschließlich über Werte aus Datenbankfeldern positioniert. Eingebundene JavaScript Funktionen werden ebenfalls in der Datenbank abgebildet. Änderungen im Code können dadurch einfacher in die Oberfläche eingepflegt werden. Code Updates und Hotfixes können auch bei individuell zusammengestellten Clientanwendungen einfach eingebunden werden.

Durch die Hinterlegung der Steuerelemente im Mapbender Datenmodell können Anwendungen wesentlich einfacher in CMS (Content Management Systeme) eingebunden werden. Dabei geht trotzdem der Charakter einer Universalapplikation nicht verloren, da Applikationen auch gänzlich ohne CMS implementiert werden können.

Implementierung dynamisch zugewiesener SLD Dokumente

Eine sehr große Bedeutung wird der seit WMS Version 1.1.1 verfügbaren Integration von

SLD (Styled Layer Descriptor) Dokumenten beigemessen. Damit besteht die Möglichkeit, in einem getMap-Request Zeichenvorschriften an den WMS Server zu übermitteln, die beim Rendering (Zeichnen) der Karte berücksichtigt werden.

Über diese Technik ist es möglich, mit standardisierter Software thematisches Mapping zu betreiben. Derzeit ist die Unterstützung von SLD durch WMS Dienste noch relativ dünn gesät. Das Open Source Projekt deegree ist allerdings die Referenzimplementierung des OGC und stellt eine sehr sauber implementierte Lösung bereit.

Erweiterte Digitalisierfunktionalität

Die Digitalisierfunktionalität wird auf Kundenanfragen hin immer weiter ausgebaut. Neben einfachem Online Digitalisieren sind bereits einige Konstruktionsfunktionen implementiert worden. Eigentlich sollte dieser Bereich nicht weiter ausgebaut werden, da die Aufgabenstellung ganz klar in Richtung Desktop GIS weist und hier bereits eine Vielzahl von Produkten zur Verfügung stehen.

Mit der immer stärker wachsenden Rolle zentraler Datenbestände wächst allerdings auch der Anspruch an Desktop GIS Anwendungen, auf diesen zentralen Datenbestand zuzugreifen. Die meisten Desktop GIS tun sich außerordentlich schwer damit, und da Mapbender bereits einfache Digitalisierfunktionalitäten enthält, wird diese zunehmend weiter ausgebaut. Dabei liegt der Schwerpunkt zunächst auf einfachen Digitalisierfunktionen, die nach und nach weiter ausgebaut werden können.

Direkte Integration der Datenhaltungs- und Pflegekomponente

Das Mapbender Projekt entwickelt sich deutlich in Richtung einer GDI Architektur, die den UMN MapServer und die PostgreSQL Datenbank mit Spracherweiterung PostGIS klammert. Durch die Integration der OGC-kompatiblen GEOS Funktionsbibliothek in PostGIS stehen über die Datenhaltungskomponente vielfältige Möglichkeiten der Datenmanipulation zur Verfügung.

Ein Schwerpunkt der Entwicklung wird deshalb die tiefere Integration der GEOS Methoden in die Mapbender Architektur sein.

Link zur Userliste

Wenn Sie Interesse an den Entwicklungen des Mapbender Projektes haben oder gerne mitmachen möchten, schreiben Sie sich in die Userliste ein. Die Userliste findet sich unter:

https://lists.sourceforge.net/lists/listinfo/mapbender-users

* * *

Beispiele für den Einsatz von WebGIS mit OS/FS

Von Till Adams & Arnulf Christl

Im folgenden Teil werden exemplarisch einige OS/FS WebGIS Lösungen vorgestellt. Immer mehr Entscheider erkennen die Vorteile von OS/FS Lösungen für ihre Fragestellungen. Besonders im aktuellen Kontext steht die Kostenfrage zunächst oft im Vordergrund, je weiter die Projekte fortschreiten, um so wichtiger werden aber auch weitere Vorteile wie z.B. ausgezeichnete Qualität und die Vorteile einer informellen aber sehr kompetenten Anwender- und Entwicklergemeinschaft.

Oft leidet die Einführung von OS/FS allerdings unter nicht besonders guter Informationspolitik. So wird zum Beispiel öfter geschrieben:

"Die lizenzfreie Open Source Lösung xy wurde..."

Das ist nicht richtig, da eine Open Source oder Freie Software Lösung nicht "unlizenziert" ist, sondern sehr klar durch eine Lizenz geschützt ist – die Nutzung dieser Lizenz aber nicht an Kosten gebunden sein muss. Korrekterweise müsste es also zumindest heissen:

"Die lizenzkostenfreie Open Source Lösung xy wurde..."

In einer weiteren Detailstufe sollte dann differenziert werden, um welche Art von Freie Software Lizenz es sich handelt, z.B. eine GPL oder L-GPL, BSD-artige oder Artistic Lizenz. Nur mit dieser Information ist ersichtlich, in welchem Kontext die Software genutzt werden kann, wenn es sich z.B. um eine L-GPL handelt (früher Library-, heute Lesser-GPL), dann können Programmteile mit proprietärer Software verlinkt und diese auch gegen Lizenzgebühr verkauft werden, nur der unter der L-GPL geschützte Teil muss offengelegt sein. Wenn es sich dagegen um eine reine GPL Lizenz handelt, dann ist diese Verlinkung mit proprietärer Software nicht erlaubt.

Aufgrund dieser speziellen Anforderungen stellen wir in diesem Bereich zunächst nur Projekte vor, von denen wir wissen, welche Bedingungen und Lizenzvorschriften Anwendung finden.

Stadt Bonn

Die Stadt Bonn betreibt seit Jahren ein sehr umfangreiches WebGIS. Ursprünglich aus einem Umweltinformationssystem (UIS) gewachsen, enthält das Bonner WebGIS inzwischen über 170 Themen. Ein Teil davon wird über den Link <Stadtplan> der Webseiten der Stadt Bonn als Internetdienst angeboten und ist Teil des Bürgerinformationssystems. Der Großteil der Themen wird den Anwendern im Intranet über eine Benutzerstruktur mit Zugriffsberechtigung für spezielle Fachschalen zur Verfügung gestellt.

Das System wird seit Jahren konsequent ausgebaut, seit zwei Jahren wird verstärkt OS/FS eingesetzt. Das Ziel ist eine vollständige, offene und freie Architektur, die

Verwaltungsstrukturen und Arbeitsabläufe flexibel abbildet, also nicht statisch ist, sondern sich dynamisch wechselnden Anforderungen anpassen kann.

Historie

In der Stadt Bonn werden historisch bedingt sehr unterschiedliche Systeme eingesetzt, diese wurden und werden von unterschiedlichen Herstellern teils als proprietäre, teils als OS/FS Lösungen betrieben. Aus den sehr unterschiedlichen Anforderungen an eine GIS Architektur (Kataster, Umwelt, Planung) und den technischen Rahmenbedingungen der letzten zehn Jahre ist eine vollständig heterogene Landschaft entstanden. Im Folgenden wird daraus lediglich die WebGIS Komponente (ursprünglich das UIS) der Stadt Bonn vorgestellt.

http://www.bonn.de/rat_verwaltung_buergerdienste/buergerdienste_online/buergerservice_a_z/00643/

Als erste Komponenten des Systems wurden Desktoparbeitsplätze eingerichtet. Um der steigenden Nachfrage innerhalb der Stadtverwaltung gerecht zu werden, wurde der Bestand nach und nach aufgestockt. Zusammen mit dem Hersteller dieser Software, dem Fachbereich Geographie der Universität Bonn und der CCGIS wurde die erste Weblösung implementiert. Teile dieser Lösung sind heute noch in Betrieb, andere wurden bereits durch neue Komponenten ersetzt. Trotz aller Änderungen konnte aber der Online Betrieb über die Jahre immer aufrechterhalten werden. Mit dem Relaunch der neuen Bonner Webseite im November 2003 gibt es parallel zu der proprietären auch eine vollständig mit OS/FS Komponenten ausgestattete WebGIS Architektur. Als erste OS/FS Komponente wurde die alphanumerische Datenhaltung (SQL Datenbank) umgestellt. Inzwischen läuft die PostgreSQL Datenbank als Primärsystem im Tagesbetrieb.

Die bisherige Clientlandschaft enthielt viele eigene Komponenten, die für spezielle Anwendungsgebiete entwickelt wurden. Diese konnten in die freie Mapbender Client Suite integriert werden, die seit 2003 in der Stadt Bonn zum Einsatz kommt. Gleichzeitig konnte Mapbender auch an OGC Standards angepasst werden, um eine weitgehend freie Architektur zu ermöglichen. Die Umstellung der Clientlandschaft auf Mapbender ermöglicht es jetzt auch, einen deegree Geodatenserver zu verwenden. Damit können die OGC Dienste WMS, WFS, WTS, WCAS aktiv genutzt werden.

Bei der Umstellung der WebGIS Anwendungen auf OS/FS spielt vor allem die Offenheit der Architekur eine wichtige Rolle, da die Stadt Bonn schon immer eine Fülle an eigenen Entwicklungen bereitgestellt hat und dies auch in Zukunft tun wird. Durch die Einhaltung von Standards und der OS/FS Lizenzpolitik ist es jetzt auch wesentlich einfacher möglich, dass andere Anwender von dem Rückfluss dieser Entwicklungen in die Basissoftware profitieren. So konnte bereits eine Integration von Mapbender in das stadtweite CMS erfolgen, das Ergebnis ist zusätzlich eine vollständige Konfigurierbarkeit der Mapbender Oberflächenkomponenten über eine Datenbank.

Gemeinsame Nutzung von Geodaten

Die Geodatenvermarktung ist in der Stadt Bonn gekennzeichnet durch die nachhaltige Inwertsetzung der Daten. Eine Inwertsetzung erfolgt durch eine Mehrtwertbildung, den die Stadt Bonn durch Aufbereitung der Geodaten und Bereitstellung als WMS Dienst mit eigenen Erweiterungen der Mapbender Client Suite erzielt. Der Dienst kann auch über Domänengrenzen hinweg genutzt werden. So konnten z.B. die Fahrplandaten der regionalen Nahverkehrsauskunft nahtlos integriert werden.

Die Zusammenarbeit zwischen den Stadtwerken und der Stadt ist hervorragend. Daher können erstere als Datenprovider z.B. die Bushaltestellen (Koordinaten) für den Stadtplan bereitstellen und im Gegenzug Kartendienste und Daten der Stadt in eigenen Anwendungen nutzen. Eine solche WebGIS Anwendung ist das Disponenten-Einsatzsystem, das gerade in eine neue Etwicklungsphase gekommen ist. Hier erfolgt eine Umstellung der Basissoftware von einer lokalen, rein Microsoft-basierten Active-X Anwendung zu einer HTTP-basierten, offenen OS/FS Lösung.

Software

Betriebssystem: SuSE Linux
Webserver: Apache HTTP Server Project
Servletcontainer: Apache Jakarta Project / Tomcat
Datenbank: PostgreSQL/PostGIS
Kartenserver: deegree WMS
Rasterkartenserver: deegree WCS
Geometrieserver: deegree WFS
Geländemodellserver: deegree WTS
Kartenclient: Mapbender Client Suite SPA

Betrieb

Die modulare Architektur ermöglicht es, den Betrieb des Systems aufzuteilen. Die Datenbanken, die Kartenserver, die Web-Server werden je nach administrativen Gesichtspunkten verteilt vorgehalten. Des weiteren werden im Rahmen der oben beschriebenen Kooperation Daten und Dienste der Stadtwerke genutzt.

Stadt Soest

Die Stadt Soest setzt in der Verwaltung ca. 12 Desktop-GIS Lizenzen ein. Seit Anfang 2004 kommt hausweit der UMN MapServer als webbasierte Kartenauskunftskomponente zum Einsatz. Wichtige Kriterien bei der Entscheidung für diese Lösung sind die gute Bedienbarkeit der Oberfläche sowie die hohe Leistung und Zugriffsgeschwindigkeit der Serverkomponenten.

Systemarchitektur

Für die Kartenauskunft wurde ein eigener Server aufgesetzt und mit einer vollständig freien WebGIS Architektur inkl. Betreibssystem und Datenbank ausgestattet. Die Serverhardware ist mit einem Standard 2.4 GhZ Prozessor und 1 GB RAM augestattet.

Software

Betriebssystem: Debian GNU/Linux
Webserver: Apache HTTP Server Project
Datenbank: MySQL
Kartenserver: UMN MapServer v4.01
Kartenclient: Mapbender Client Suite

Anwendung

Über die Mapbender Client-Suite werden derzeit 5 thematisch gegliederte Kartendienste bereitgestellt. Als Geobasisdaten werden die DGK5, Luftbilder und weitere Ebenen eingebunden. Die B-Plan-Auskunft ermöglicht entweder über eine Suchfunktion oder die Umringgeometrien in der Karte wahlfreien Zugriff auf B-Pläne. Diese liegen nicht georeferenziert vor, sondern können über eine Abfrage als PDF-Dokument geöffnet werden.

Weiterhin sind Daten über Höhenpunkte, das Baumkataster, Natur- und Landschaftsschutzgebiete, Sozialeinrichtungen, Spielplätze und Verkehr abrufbar.

Der UMN MapServer greift auf den gleichen Geodatenbestand zu wie die Desktop-Arbeitsplätze. Dies bedeutet, dass von der AG Geoservice bearbeitete Geodaten direkt hausweit allen Nutzern zur Verfügung stehen.

Weitere Ausbaustufen

In einem nächsten Schritt wird die dateibasierte Datenhaltung auf eine datenbankbasierte Umgebung portiert. Durch die Zusammenführung der Geometrie und Alphanumerik in eine zentrale Instanz entsteht ein Datenbestand, der für allen Nutzern über eine klar definierte Berechtigungsstruktur bereitgestellt wird. In der gemeinsamen Datenhaltung können gleichzeitig Editiersitzungen gestartet werden. Ein Transaktionsmanagement

gewährleistet die Datenkonsistenz.

Niedersächsisches Landesamt für Straßenbau

Im Landesamt für Straßenbau (NLSTB) ist ein browserbasiertes, interaktives Auskunftssystem für die Darstellung und das Monitoring aller landesweit vorhandenen Autobahn-Tankstellen und -Rastplätze implementiert worden.

Daten

Die Daten des Systems wurden in den letzten Jahren mit der GIS-Fachschale "Tank- und Rastplatzanlagen" erstellt und gepflegt. Mit den neuen WebGIS Komponenten werden diese Daten jetzt über das Intra- und Internet einer wesentlich breiteren Anwendergruppe bereitgestellt.

Zusätzlich zu den vorhandenen Geoinformationen wurden Attribute aus einer OKSTRA-konformen Datenbank (OKSTRA = Objekt Katalog Straße) verknüpft.

Architektur

Die Architektur nutzt die vom OGC spezifizierte WMS Schnittstelle als Bindeglied zwischen Kartenclient und -server. Damit ist eine weitgehende Unabhängigkeit zwischen Applikationsschicht und Serverbereich gewährleistet. Die Vorgaben des ISO Standardisierungsgremiums werden bei der Erstellung von Metainformationen als Grundlage herangezogen.

Die Architektur basiert auf einem W2K Server auf dem ein UMN MapServer v4.01 und die Mapbender Client Suite installiert sind.

Daten

Das gesamte Kartenwerk besteht aus 227 Detailplänen der Rastplätze, davon sind 46 bewirtschaftet. In den Detailplänen werden bis zu 1000 Objektklassen verwaltet, die über ca. 30 verschiedene Themen gegliedert sind. Damit wird eine Orientierung in dem sehr umfangreichen Datenbestand möglich.

Funktionalität

Die Anwender können über verschiedene Werkzeuge und Methoden zu jedem beliebigen Objekt navigieren. Über eine Übersichtskarte des Landes Niedersachsen kann jede Tank- und Rastanlage visualisiert und mit dem jeweiligen Detailplan in eine eigene Oberfläche geladen werden. In dem Übersichtsplan können bereits die Betriebsdaten zu allen

Anlagen direkt abgefragt und der Detailplan geöffnet werden. Neben der Kartennavigation ermöglicht eine Suchfunktion über Namen, Zuständigkeitsbereiche, Straßenmeistereien oder weitere Attribute die direkte Navigation zu einzelnen Tank und Rastanlagen.

Auf den Detailplänen können zu bestimmten Objekten, wie z.B: Parkplätzen, Laternen, Straßenschildern etc. Daten abgefragt werden. Die Abfrage erfolgt über die in der OGC WMS Spezifikation definierte Methode "getFeatureInfoRequest". Der UMN MapServer liefert in dem Ergebnis die Tank- und Rastanlagen ID zurück, die serverseitig in einen SQL-String eingebunden wird und eine externe Datenbank abfragt.

In einer nächsten Ausbaustufe werden weitere OGC konforme WMS eingebunden, um Daten des NWSib (Straßeninformationsbank Nordrhein-Westfalen) in das System zu integrieren.

Stadt Wesseling

Die Stadt Wesseling betreibt ein WebGIS auf Basis von OS/FS Technologie sowie mehrere Desktop Arbeitsplätze eines proprietären GIS.

Ausgangssituation / Voraussetzungen

Die Stadt Wesseling hat 1995 die ersten Lizenzen für ein proprietäres GIS erworben, um geodatenbezogene Fragestellungen zu bearbeiten. Für diese Plattform wurden inzwischen einige Fachschalen implementiert, bzw. Lizenzen (Nutzungsrechte) einiger fertiger Lösungen erworben. Um die Geodaten einer breiteren Anwendergruppe innerhalb der Stadtverwaltung zur Verfügung zu stellen, wurde und wird eine WebGIS Lösung aufgebaut. Zu der weiteren Zielgruppe, die dieses WebGIS nutzen sollen, gehören auch die Bürger. Ihnen sollen mit einem Informationssystem relevante Daten interaktiv in einem Kartenwerk zur Verfügung gestellt werden.

Die Stadt Wesseling hat erkannt, dass Interoperabilität und Offenheit die Grundsteine einer soliden, sicheren Investition in ein Geographisches Informationssystem sind. Diese Forderungen harmonieren sehr gut mit OS/FS Konzepten und erleichtern die Entscheidung für ein ganz auf offenen Technologien basierendes Gesamtsystems. Eine schlagartige Umstellung des gesamten Systems ist nicht möglich und nicht erforderlich, alle neuen Komponenten in der Architektur werden aber auf die OS/FS Konzepte ausgerichtet. Damit hat die Stadt Wesseling die Kontrolle über die weitere Entwicklung in der Hand.

Software

Betriebssystem: SuSE 9
Webserver: Apache HTTP Server Project
Datenbank: PostgreSQL/PostGIS, MySQL
Kartenserver: UMN MapServer
Kartenclient: Mapbender Client Suite SPA
Datenimport: Mapbender PostGIS Migrator

Daten

Die erste digitale Karte im WebGIS ist der Stadtplan. Dieser wurde in den letzten Jahren auf Basis der DGK5, ATKIS Daten, ALK Daten und eigener Erhebungen eigenständig mit dem proprietären Desktop GIS erstellt. Zusätzlich können Geobasisdaten des Landesvermessungsamtes NRW eingebunden werden, dazu zählen die blattschnittfreie DGK5 und Luftbilder. Die Zusammenarbeit mit den Stadtwerken, die ein eigenes proprietäres GIS haben, wurde ausgebaut, da dort digitale Daten vorliegen, die in der Stadtverwaltung benötigt werden und umgekehrt (siehe Stadt Bonn).

Funktionalität

Die Basisfunktionalitäten eines WebGIS beinhalten eine Suchkomponente, Navigationsmöglichkeiten in der Karte, die Steuerung der Karteninhalte und einen Ergebnisbereich, um Daten anzuzeigen und weiter zu verarbeiten. Diese Basis-funktionalität reicht für eine Verwaltungs- und Bürgerinformation völlig aus, deshalb ist eines der ersten Ergebnisse des WebGIS ein interaktiver Stadtplandienst.

Ziele

Das WebGIS wird eingesetzt, um der gesamten Verwaltung den Zugriff auf Geodaten und -dienste zu ermöglichen. Nach und nach sollen dann neue Fachschalen für spezielle Fragestellungen eingebunden oder implementiert werden. Parallel dazu werden die stadteigenen Geodaten weiterhin mit einem Teil der Desktop Lizenzarbeitsplätze gepflegt (z.B. die kleinräumige Gliederung), aufbereitet (z.B. in Form eines Stadtplans) und neue Daten (z.B. Gewerbeflächen, Bauleitplanung) erhoben. Ob oder wann diese Desktop Arbeitsbereiche auch vollständig durch WebGIS Technologie abgelöst werden ist noch nicht abzusehen, diese Entscheidung ist derzeit auch nicht erforderlich, da das aktuelle System zufriedenstellend läuft, wenig laufende Kosten verursacht und alle neuen Komponenten kompatibel und offen sind.

LIZ Blattkrankheiten-Monitoring

Der Landwirtschaftliche Informationsdienst Zuckerrübe (LIZ, eine Beratungsorganisation der Zuckerunternehmen: Pfeifer & Langen, Diamant-Zucker KG, Zuckerfabrik Jülich AG, Nordzucker AG und Danisco Sugar GmbH) führt seit dem Jahr 2000 jährlich ein Blattkrankheiten-Monitoring für Zuckerrüben durch. Dabei werden ca. 300 Rübenparzellen in West- Nord- und Ostdeutschland wöchentlich auf den Befall verschiedener Blattkrankheiten untersucht. Die Ergebnisse werden in Form von zoombaren Karten, standortübergreifenden Tabellen und Grafiken zum Befallsverlauf online zur Verfügung gestellt. Die Eingabe der Daten erfolgt ebenfalls online.

Internet: http://www.liz-online.de/
Folgen Sie hier den Links Pflanzenschutz -> Blattkrankheiten -> Monitoring
Starten Sie den Dienst direkt auf http://wms.ccgis.de/liz_9/index.html

Ausgangssituation / Voraussetzungen

Bis zur Einführung des GIS-Systems wurde vom Auftraggeber noch keine geographische Datenverarbeitung betrieben. Es lagen bisher auch keine Geodatengeometrien vor, lediglich eine Adresse und Ortsbezeichnung für den Ackerschlag. Mit diesen Daten wurde mittels einer geographischen Software (Geoinformationssystem) wöchentlich eine aktuelle, statische Karte erstellt.

Ziel

Nutzung von GIS Technologie für die Erfassung, Verwaltung und online-Darstellung aktueller Daten.

Das System wurde nicht nur vollständig mit Freier Software implementiert, sondern enthält ausserdem echte freie Daten. Die Daten wurden von einem Internet Server heruntergeladen. Sie haben lediglich eine grobe Auflösung und sind für eine echte Verortung von Ackerschlägen nicht geeignet. Die Qualität reicht für die erforderlichen Arbeitsschritte jedoch vollkommen aus. Falls das System ausgebaut wird und genauere Daten benötigt werden, so können diese jederzeit in einem Standardformat hinzugeladen werden. Dabei können auch einzelne, ausgewählte Gebiete (z.B. Luftbildaufnahmen von Ackerschlägen) eingebunden werden.

Software

Betriebssystem: freeBSD, Windows
Webserver: Apache HTTP Server Project, IIS 5.0

Datenbank: PostgreSQL/PostGIS, MySQL
Kartenserver: UMN MapServer (über WMS Schnittstelle)
Kartenclient: Mapbender Sugar (mit Java Applet)

Daten

World Data; Country Germany (USA NCAP Download)
Eigene Erhebung (in Excel-Format)

Betrieb

Das System wird vollständig gehostet, der Anwender nutzt die Applikation als ASP System. Die Hard- und Software ist beim Dienstleister installiert, dort erfolgt auch die tägliche Datensicherung. Bei Bedarf werden Hotfixes eingespielt, die Verfügbarkeit des Dienstes wird gewährleistet und eine Hotline für die Anwender wird ebenfalls betrieben.

Stadt Bielefeld

Die Stadt Bielefeld betreibt ein WebGIS auf Basis proprietärer und OS/FS Technologie. GIS Desktop Arbeitsplätze, Fachanwendungen, Workstations und Datenbankserver bilden dabei das GIS Basissystem. Parallel dazu wird im Internet ein einfacher Stadtplan betrieben, der exakt auf die Anforderungen eines Stadtinformationssystems abgestimmt ist.

Die neue WebGIS Lösung wird zunächst genutzt, um die Eigenentwicklung abzulösen, da sie sowohl von der Leistung als auch der Funktionalität an ihre Kapazitätsgrenzen stößt. Die höhere Leistungsfähigkeit des neuen Systems ermöglicht u.a. die Anbindung von wesentlich mehr verwaltungsinternen Anwendern als bisher. Externen Anwendern kann über einen kontrollierten Zugang der Zugriff auf die Geodienste freigeschaltet werden. Dabei können alle Bewegungen und Anfragen protokolliert werden.

Funktionalität

Eine der ersten zentralen Aufgaben des neuen WebGIS ist der Zugriff auf Eigentümernachweise über die Kartenkomponente. Die erforderlichen Basisdaten werden duch die bestehenden Systeme bereitgestellt und über Standardschnittstellen verknüpft (OGC WMS). Die Erweiterung des Systems erfolgt durch den Einsatz neuer Technologien, die parallel zu der bestehenden GIS Landschaft mit kleinen Schritten in Betrieb genommen werden kann.

Software

Betriebssystem: SuSE Linux, Windows 2000 Server
Webserver: Apache HTTP Server Project, Internet Information Server 6.0
Datenbank: PostgreSQL/PostGIS, MySQL, MS Access
Kartenserver: Proprietärer IMS mit WMS Unterstützung, UMN MapServer mit WMS
Kartenclient: Mapbender Client Suite SPA / Outsider
Datenimport: Mapbender PostGIS Migrator

Software-Mix

Der Software-Mix aus proprietären und OS/FS Paketen zieht sich durch die gesamte Architektur des WebGIS. Die einzelnen Komponenten werden dabei, soweit dies möglich ist, über Standardschnittstellen miteinander verbunden. Durch das in der Stadt Bielefeld vorhandene Know-How können die Verantwortlichen bei Bedarf erforderliche und gewünschte Komponenten und Schnittstellen in Eigenregie implementieren. Für die Kernstücke der Architektur jedoch (sozusagen die Motoren und Getriebe) wird der Einsatz von OS/FS-Basissoftware angestrebt, da eine Eigenentwicklung dieser Hochleistungs-Serversoftware nicht geleistet werden kann. OS/FS Lösungen bieten hier die beste Möglichkeit, gleichzeitig ein Standardprodukt einzusetzen und trotzdem eng an der Weiterentwicklung mitzuwirken.

*** * ***

Schluss

Von Arnulf Christl & Athina Trakas

Am Schluss angelangt, sollen an dieser Stelle nun nochmals einige "Fäden zusammengeführt" und einige der nicht so offensichtlichen Vorteile von Freier und Open Source Software anfgeführt werden.

Wie im Laufe des Handbuchs dargestellt, mit praktischen Beiträgen unterfüttert und mit Beispielen für den möglichen Einsatz von Freier Software in verschiedenen Bereichen näher erläutert wurde, gibt es für WebGIS Systeme ausreichend Produkte und Tools, die als Freie Software zur Verfügung stehen. Von unterschiedlichen Betriebssystemen und Datenbanken über WMS Dienste bis hin zu Clients aber auch Werkzeugen wie AveiN!, ist das Spektrum umfassend – es ist für jede Fragestellung etwas dabei. Auch Architekturen im Software-Mix (freie und proprietäre Software) sind möglich und in manchen Bereichen sicherlich sinnvoll.

WebGIS mit OS/FS Ansatz ist nicht nur technisch möglich, sondern wird bereits produktiv praktiziert.

- Die technischen Anforderungen an eine WebGIS Architektur sind mit OS/FS Technologie lösbar.
- Die offene Lizenzpolitik von OS/FS fördert die einfache Multiplizierung dieser Lösungen und führt zu einer schnell wachsenden Anwendergemeinschaft.
- Das höhere gemeinsame personelle und finanzielle Potential dieser Anwendergemeinschaft kann die Verbesserung und Entwicklung neuer Software beschleunigen.
- Die Integration der wachsenden Anwenderzahl verbessert die Qualität der Software durch umfangreiche Anwendung in der Realität, zusätzlich zu Tests im Labor.

Gerade im Bereich GIS und WebGIS ist ein sofortiger, schlagartiger Wechsel von einer Softwareideologie in eine andere weder sinnvoll noch erforderlich. OS/FS Entwicklungen bieten für die speziellen Fragestellungen einer WebGIS Infrastruktur ideale Rahmenbedingungen und der Anteil an lauffähigen und produktiven WebGIS Architekturen mit FS/OS steigt täglich.

Einige der Vorteile des Einsatzes von OS/FS wurden im einen oder anderen Kapitel bereits angeschnitten. Einige andere sollen an dieser Stelle noch genannt werden.

Neben des in Zeiten leerer Kassen immer wieder gerne bemühten Arguments der Kosten, gibt es noch eine Vielzahl an weiteren Aspekten, die den "Charme" Freier Software ausmachen. Die folgende Aufzählung nennt neben den Kosten noch weitere wichtige Aspekte, die für den Einsatz Freier Software sprechen.

Kosten

Kosten entstehen beim Einsatz von OS/FS ausschließlich durch die Lösung eines Problems, d.h. die Installation und Konfiguration von Softwarepaketen sowie durch Anpassungen, sofern diese an externe Stellen abgegeben werden. Es entstehen keine Kosten durch die Anschaffung von Softwarenutzungsrechten. Derzeit wird eine offene und z.T. heftige Diskussion um diese Kosten geführt.

Die gesammelte Erfahrung aller hier vorgestellten Projekte und aller beteiligten Autoren hat gezeigt, dass die technischen Probleme - also die Beherrschung einer Technik - immer lösbar sind, egal ob es sich dabei um proprietäre oder um Freie Software handelt. Das Know-How für den Aufbau einer Geodateninfrastruktur muss in beiden Fällen aufgebaut werden und die Kosten dafür hängen von so vielen Faktoren ab, dass eine Aussage, ob ein auf OS/FS basierendes oder proprietäres System besser abschneidet, nur individuell entschieden werden kann.

Die potentielle Kostenersparnis ist sicherlich ein Argument, das besonders in finanziell schwierigen Zeiten ein gewichtiger Faktor bei der Entscheidung für einen OS/FS Ansatz ist. Die zur Verfügung stehenden Mittel können dabei vollständig in konkrete Problemlösungen investiert werden, und werden nicht zusätzlich durch proprietäre Lizenznutzungsrechte reduziert.

Nutzen

Der funktionale Nutzen eines Systems schlägt sich in Zugriffszahlen nieder, die zusammen mit der Zufriedenheit der Anwender eine qualitative Aussage ermöglichen. Ob die Effektivität der Arbeitsabläufe für die Anwender durch das System steigt, ist schwer messbar, Auskunft darüber können nur die Anwender selbst geben. Wenn also überhaupt irgendein Index entwickelt werden kann, an dem der Nutzen eines Systems erkennbar ist, dann nur indem man den Anwender als zentralen Beitragenden in das Bewertungssystem miteinbezieht.

Der finanzielle Nutzen eines Systems lässt sich durch eine Investition (Soll) und das darauf folgende Return on Investment (Haben) beziffern. Die Erfahrungen mit kommerziell operierenden Firmen haben gezeigt, dass der Einsatz von OS/FS dort immer ein sehr gutes Verhältnis von Investition und Rücklauf ergibt.

Wenn kein vollständig marktabhängiger, kommerzieller Apparat zur Verfügung steht wie z.B. in der öffentlichen Verwaltung, ist es schwieriger, diesen Nutzen zu quantifizieren. Der Rücklauf auf Investitionen in eine WebGIS Architektur wirkt immer nur indirekt, beispielsweise durch qualitativ hochwertigere Entscheidungen bei der Stadtplanung. Ob die Entscheidungen wirklich qualitativ hochwertiger sind kann in Ermangelung einer Regelmäßigkeit und Vergleichbarkeit weniger gut beurteilt werden.

Effizienz

Die Effizienz von OS/FS Lösungen kann sehr hoch sein. OS/FS Technologie kann eingesetzt werden, um zentral vorgegebene und strukturierte Top-Down Architekturen umzusetzen.

Um eine möglichst hohe Effizienz von OS/FS Entwicklungskonzepten zu erzielen, sollte der Ansatz aber immer die Lösung konkreter Probleme sein - dann entfaltet OS/FS erst ihr volles Potential.

Die Einhaltung von Standards ist bei Geodateninfrastrukturen ein Muss, egal ob bei proprietären, freien oder Mischarchitekturen.

Qualität und Sicherheit

Mehrere Studien (z.B. Bundesamt für Sicherheit in der Informationstechnik - http://www.bsi.de, bundestux – http://www.bundestux.de) kommen zu der Erkenntnis, dass Open Source und Freie Software von hoher Qualität ist und bei Tests sehr gute Ergebnisse erzielt. Dies ist für Nutzer und Betreiber von großem Vorteil, da die Systeme und Dienste ein hohes Maß an Zuverlässigkeit erreichen.

Das Argument, dass zugängliche Quellcodes gleichzusetzen sind mit Sicherheitslücken, ist nicht zutreffend. OS/FS Konzepte widersprechen Sicherheitsaspekten nicht, sondern unterstützen sie. Das sog. Linus Law beschreibt paralleles debugging, das erst durch den Zugang zum Quellcode möglich ist: viele Programmierer sehen mehr als wenige, Fehlerquellen und potentielle Sicherheitslücken werden schnell erkannt und können direkt und effektiv korrigiert werden. Dadurch steigt das Sicherheitsniveau von Freier Software stetig an.

Digitale Dauerhaftigkeit von Daten

Der eigentliche Wert einer Geodateninfrastruktur liegt in den Daten, nicht in der Software.

Digitale Daten müssen unter Umständen sehr lange zur Verfügung stehen. Dies betrifft insbesondere auch Geo-Daten, die sich über Jahre hinweg nur teilweise ändern oder die für eine Historienverwaltung archiviert werden müssen. Wichtig bei der Archivierung ist, dass Daten, die von Hard- und Software gespeichert wurden, die möglicherweise längst nicht mehr auf dem Markt verfügbar ist, auch nach einem längeren Zeitraum wieder dekodiert werden können.

Ein gutes Beispiel sind Satellitenbilder aus den ersten Jahren der Raumfahrt, die historische Daten enthalten, aber leider nicht mehr lesbar sind. Dieser Bereich der „Software und Daten-Archäologie“ wird in Zukunft eine immer größere Bedeutung erhalten. Die Offenlegung des Quellcodes der Software, mit der Daten ursprünglich erhoben und physikalisch gespeichert wurden, gewährleistet, dass die Daten auch nach Jahren noch lesbar sind.

Hohes Potential für gemeinsame Lösungsansätze

Wie bereits beschrieben, haben gerade öffentliche Verwaltungen sowohl ähnliche Fragestellungen als auch die gleichen Lösungsansätze. Um diese zu bündeln, bieten sich OS/FS Konzepte idealtypisch an (z.B. stadt-, kreis- und landesübergreifende Nutzung von Geodaten und Methoden).

Durch das gemeinsame Lösen von Fragestellungen können Kosten und Zeit gespart werden. Beim Einsatz von Freier Software, die den Zugang zum Quellcode ermöglicht, ist eine gemeinsame Entwicklung an der Software möglich. Hier können die Synergieeffekte der Zusammenarbeit und der Nutzung des Wissens innerhalb einer Anwender-gemeinschaft direkt umgesetzt werden. In eben dieser gemeinsamen Umsetzung liegt ein enormes Potential, das insbesondere auch im Bereich GIS und Geodaten von großem Vorteil ist.

*** * * * ***

Impressum

Es haben mitgewirkt

Till ADAMS

→ WebGIS & UMN MapServer, AveiN!, Beispiele

Christina BIAKOWSKI

→ PostgreSQL/PostGIS, UMN als WMS

Arnulf CHRISTL

→ Vorwort & Einleitung , GDI Architekturen, OGC & WMS, Mapbender, Beispiele, Fazit

Astrid EMDE

→ OGC & WMS

Benjamin THELEN

→ Plattformen

Athina TRAKAS

→ Projektorganisation, Vorwort & Einleitung, Fazit, Layout

Wir bedanken uns bei den vielen Kunden, Anwendern und Entwicklern, die durch Hinweise und Beiträge zu diesem umfangreichen und praxisbezogenen Leitfaden beigetragen haben und hoffen, dass auch Sie dieses Projekt weiter mit uns ausbauen.