Mobile Geodatenvisualisierung mit Android

Mathias Menninghaus

Institut für Geoinformatik und Fernerkundung
Universität Osnabrück
Barbarastr. 22b, 49076 Osnabrück
mmenning@uni-osnabrueck.de

Abstract: Mit der zunehmenden Leistungsfähigkeit mobiler Endgeräte wird es immer reizvoller, Desktopanwendungen mobil verfügbar zu machen. Nicht nur der ständig verfügbare Netzzugriff, sondern vor allem die steigende Rechengeschwindigkeit, GPS Empfänger, Beschleunigungssensoren und Kameras in heutigen Smartphones machen es möglich, Daten überall standortbezogen abzufragen. Während ein Geoinformationssystem (GIS) im herkömmlichen Sinn hohe Rechenkapazität für immer komplexere Werkzeuge fordert und enorme Datenmengen bearbeitet, macht es besonders die Arbeit mit georeferenzierten Daten oft erforderlich, sich „vor Ort“ ein Bild zu machen.
Als erster Schritt in Richtung eines freien mobilen Geoinformationssystems wurde im „Studienprojekt Geoinformatik“ an der Universität Osnabrück eine Applikation für Googles mobiles Open-Source-Betriebsystem Android entwickelt, auf deren Basis die Eignung verschiedener Ansätze untersucht wird. Zu Beginn der Betrachtung steht die Darstellung zweidimensionaler Geodaten mit Hilfe der standardisierten Schnittstellen Web Map Service (WMS) für Rasterdaten und Kartenmaterial sowie Sensor Observation Service (SOS) für raumbezogene Sensordaten. Weiterhin werden *Really Simple Syndication* (RSS) Feeds mit Raumbezug (GeoRSS) genutzt. An ihnen wird überprüft, wie der Endnutzer über raumbezogene Ereignisse, wie diese beispielsweise bei Frühwarnsystemen gegen Naturgefahren benötigt werden, informiert werden kann. Zur Beurteilung der potentiellen Leistungsfähigkeit mobiler Geoinformationssysteme werden am Ende der Betrachtung dreidimensionale geologische Daten mit OpenGL ES auf dem mobilen Gerät visualisiert. Den speziellen Anforderungen der Applikationsentwicklung auf mobilen Engeräten wird dabei Rechnung getragen. Die Applikation soll schließlich im ersten Quartal 2010 als freie Software veröffentlicht werden.

1 Einleitung

Mit der zunehmenden Leistungsfähigkeit mobiler Endgeräte wird es immer reizvoller Desktopanwendungen mobil verfügbar zu machen. Nicht nur der ständig verfügbare Netzzugriff, sondern vor allem die steigende Rechengeschwindigkeit, GPS Empfänger, Beschleunigungssensoren und Kameras in heutigen Smartphones machen es möglich, Daten überall standortbezogen abzufragen.
Ein Geoinformationssystem (GIS) [Bar05, S.15ff] fordert hohe Rechenkapazität für im-

mer komplexere Werkzeuge und bearbeitet enorme Datenmengen. Doch besonders die Arbeit mit georeferenzierten Daten macht es oft erforderlich, sich „vor Ort“ ein Bild zu machen. So genau die erhobenen Daten auch sein mögen, sie geben immer nur einen Ausschnitt der Realität wieder. Zudem eignet sich ein mobiles Gerät ideal als Träger für einen Frühwarnsystem-Klienten zum Beispiel im Rahmen der Entwicklung von Frühwarnsystemen gegen Naturgefahren [B+07]. Genau deswegen ist es sinnvoll, ein System zu entwickeln, mit denen georeferenzierte Daten direkt im Gelände abgerufen und geeignet dargestellt werden können. Hard- und Software grenzen ein solches Vorhaben jedoch schnell ein, denn nicht überall ist der Netzempfang ideal und trotz gestiegener Rechenkapazität können Smartphones nicht die Leistung von Desktop-PCs oder gar Servern mit GIS Funktionalität bieten. Dieser Problematik soll von zwei Seiten entgegengegangen werden.
Erstens synchronisiert sich die Anwendung solange wie Netzzugriff besteht und ist dabei aber in der Lage, bereits synchronisierte Daten auf dem Gerät für eine spätere Verwendung wieder vorzuhalten. Außerdem sollte es möglich sein, die Daten von vornherein auf das Endgerät zu übertragen und so den fehlenden Netzzugriff ignorieren zu können. Dabei ist besonders auf eine geschickte Auswahl der relevanten Daten und die Speicherkapazität zu achten.
Zweitens übernimmt an den Stellen, an denen das mobile Endgerät die nötige Leistung nicht bringen kann, ein Server, der auf GIS spezifische Abfragen optimiert ist, die Berechnung. Ausschlaggebend hierfür ist eine effiziente Server-Client-Struktur, die schnelle Abfragen ermöglicht.

1.1 Die verwendeten Standards

Die Formate und Services mit denen Geodaten abgefragt werden, sollen möglichst standardisiert sein um ein breiteres Anwendungsgebiet zu ermöglichen. Es geht nicht darum, einen speziellen Datensatz zu unterstützen, sondern dem Benutzer die Möglichkeit zu geben, ihm bekannte oder gar selbst aufgesetzte standardkonforme Services mobil zu nutzen. Deswegen werden für die Visualisierung im 2D-Teil die Standards Web Map Service (WMS) [dLB02] für die Darstellung von Kartenmaterial, Sensor Observation Service (SOS) [NP07] für die Abfrage von Sensordaten und Geo-enabling RSS Feeds (GeoRSS) [Ree06] für die Darstellung ereignisbezogener Punkdaten verwendet. Sie alle werden vom Open Geospatial Consortium (OGC) [ogc10] betreut und sind frei verfügbar. Im 3D-Teil soll das GOCAD-ASCII-Datenformat [Pro10] unterstützt werden, da in Zukunft Anfragen an die 3D/4D-Geodatenbank DB4GeO [B+09] ermöglicht werden sollen, welche geologische 3D-Untergrundmodelle verwaltet.

2 Applikationsentwicklung auf mobilen Endgeräten

Zu Beginn musste entschieden werden, welches Betriebssystem für die geforderten Aufgaben am besten geeignet ist. Es stand eine große Auswahl von Smartphone-Betriebssystemen (Palm WebOS [Pal10], iPhone OS [Inc10], Windows Mobile [Mic10], Symbian OS [Fou10],

Maemo [Com10] und Android [and10a]) zur Verfügung. Zunächst sollten mobile Geräte mit diesem Betriebssystem im Einzelhandel erhältlich sein und zudem GPS Empfänger und eine gewisse Rechenleistung besitzen. Dies ist bei den meisten Smartphones der Fall und für jedes vorgeschlagene Betriebssystem lässt sich ein solches Gerät finden. Da aber in der Applikation hauptsächlich freie Software und OGC Standards verwendet werden, sollte auch das Betriebssystem des mobilen Gerätes größtenteils aus freier Open-Source-Software bestehen. Womit die zum großen Teil proprietären Betriebssyteme Windows Mobile, iPhone OS, PalmWeb OS und bisher auch Symbian OS ausscheiden. Bei der Wahl zwischen Maemo und Android überzeugt Android durch eine größere Entwicklergemeinschaft und damit besseren Support, sowie eine größere Auswahl an bisher verfügbaren Geräten.

2.1 Android

Android ist ein von der Open Handset Alliance [ope10] entwickeltes Betriebsystem für mobile Endgeräte wie Smartphones, PDAs oder Netbooks. Es ist zum größten Teil freie Software und quelloffen [and10a].

Abbildung 1: Android Systemarchitektur [and10b]

Abbildung 1 zeigt die Systemarchitektur von Android. Anhand der Abbildung wird deutlich, wie umfangreich und erweiterbar das mobile Betriebsystem ist. Die Implementierte Applikation befindet sich in der obersten Systemschicht (*Applications*) und besteht in erster Linie aus den sogenannten *Activities*. Eine *Activity* beschreibt den gesamten sichtbaren Teil einer Applikation zu einem bestimmten Zeitpunkt, wie etwa ein Anruf- oder Brow-

serfenster.
Die genutzten Komponenten aus dem *Application Framework* werden im Folgenden vorgestellt. Der *Activity Manager* dient zur Verwaltung des Lebenszyklus der Applikation. Die Komponenten *Resource* und *Package Manager* werden zur Verwaltung persistenter Daten wie Lokalisierungen und Einstellungen sowie der Quellcodedateien benutzt. Die Komponenten *Window Manager* und *View System* finden zur geeigneten Darstellung der Daten und der Benutzeroberfläche Verwendung. Dabei soll sich das Look-and-Feel der Applikation an das des Betriebssystems anlehnen. Der *Location Manager* wird zur Standortabfrage hinzugezogen. Bisher weitgehend ungenutzt ist die Komponente *Content Provider* zur Speicherung der Geo- und Metadaten. Auf die Komponente *Notification Manager* wurde ebenfalls bislang verzichtet. Diese soll aber eingesetzt werden, wenn die Applikation sich völlig von alleine im Hintergrund synchronisiert und dazu noch Frühwarnfunktionen übernehmen soll. Außen vor steht die Komponente *Telephony Manager*, denn die eventuell auf den Geräten verfügbaren Telefoniefunktionen sollen hier nicht betrachtet werden.
Die wichtigsten Bibliotheken (*Libraries*) sind, neben den Java und Android Standardbibliotheken, die Bibliothek *SQLite* zur Speicherung jeglicher Daten aus dem 2D-Bereich und *Open GL ES* zur dreidimensionalen Darstellung von geologischen Bodenschichten.
Die Laufzeitmaschine, auf der die Java-Applikation schließlich ausgeführt wird, ist die *Dalvik Virtual Machine*. Sie begrenzt den verwendeten Speicher pro Applikation auf etwa 15 MB. Diese Problematik wird später näher erläutert.
Der Linux-Kernel des Betriebssystems sorgt schließlich für die nötige Einbindung von Hardwarekomponenten. Viele dieser Hardwarekomponenten sind jedoch optional. Für die Implementation wurde zumindest von einem GPS-Empfänger und einem Flash-Speicher (SD-Karte) ausgegangen und danach auch das Testgerät ausgewählt.
Die Entwicklung von Android Applikationen erfolgt hauptsächlich mit der Entwicklungsumgebung Eclipse [ecl10] und entsprechenden Plugins, wie beispielsweise dem Android Development Tools (ADT)-Plugin zur Entwicklung von Android-Applikationen. Zum Testen der Applikationen kann ein in dem ADT-Plugin enthaltener Emulator verwendet werden, der wie ein echtes Gerät über die Android Debug Bridge (ADB) mit einem PC verbunden wird.

2.2 Informationsdarstellung

Smartphones oder ähnliche Geräte besitzen hauptsächlich Bildschirme mit Größen bis 3.2 Zoll Bildschirmdiagonale und einer Auflösung von 320 mal 480 Pixeln. So gilt es, dort Inhalte möglichst kompakt darzustellen, um häufige Scroll-Operationen zu vermeiden, dabei aber nicht den Informationsgehalt der Darstellung zu mindern.

Die besprochenen Geräte besitzen ständigen Internetzugriff, hauptsächlich über UMTS oder W-LAN Netzwerke. Doch UMTS und W-LAN Netze sind mitnichten flächendeckend, allenfalls in Großstädten. Von Vetriebsseite wird oft das *Anywhere Anytime Computing* [MS04] angepriesen, in der Realität muss aber mit häufigen Verbindungsproblemen gerechnet werden. Dieser Problematik muss der Entwickler Rechnung tragen. Dabei ist es keine Lösung von vornherein alle Daten auch offline auf dem Gerät vorzuhalten und bei-

zeiten zu synchronisieren, da dies die begrenzten Speicherkapazitäten des mobilen Gerätes sprengt.
Zusammenfassend muss man bei der Entwicklung von mobilen Applikationen die Erwartung des Benutzers von *Anywhere, Anytime Computing*, den unzuverlässigen Netzzugriff und die begrenzten Systemressourcen beachten, um ein zufriedenstellendes Ergebnis zu erzielen.

3 Mobile Beispielapplikation *GeoViewer*

Abbildung 2 zeigt das vereinfachte Schema der implementierten Applikation *GeoViewer*. Auf die Darstellung der Paket- und Klassenstruktur wurde dabei verzichtet, es soll vielmehr ein Überblick darüber gegeben werden, wie die einzelnen Funktionen in die Gesamtapplikation eingebunden wurden. Unter [Men10] wird der gesamte Quellcode frei zur Verfügung gestellt.

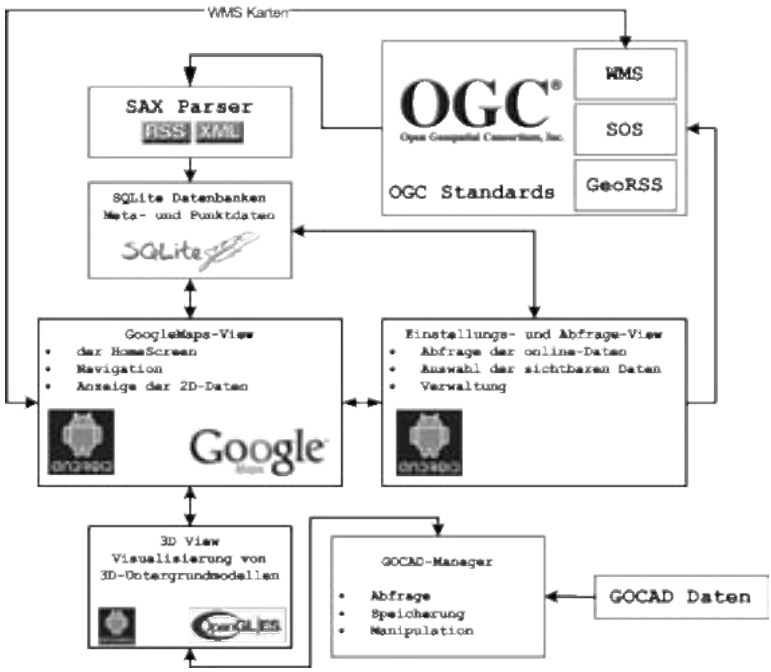

Abbildung 2: Vereinfachter Applikationsentwurf

Der Kern der Applikation ist die Komponente *GoogleMaps-View*. In dieser werden alle Informationen gebündelt und auf einer Karte mit Hilfe der GoogleMaps-API visualisiert. Die OGC-Services werden über weitere Abfrage- und Einstellungsansichten verwaltet. In die-

sen erfolgt u.a. die Auswahl der darzustellenden Komponenten und deren grundsätzliches Aussehen (Farbe, Größe usw.). Die Abfrage von Metadaten, den sogenannten *Capabilities*, wird hier ausgeführt und richtet sich direkt an die Services (*OGC-Standards*). Die vom Server zurückgelieferten XML Dateien werden mit Hilfe eines SAX-Parsers [sax10] in ein Datenmodell übertragen und in die auf dem mobilen Gerät verfügbaren SQLite-Datenbanken [sql10] geschrieben. Da die Services modular implementiert wurden und austauschbar bleiben sollen, referiert jeder zu einer eigenen mobilen SQLite Datenbank. Von diesen Datenbanken kann dann wiederum die *GoogleMaps-View* ihre Daten beziehen und die entsprechenden *Einstellungs- und Abfrage-Views* können diese Daten zusätzlich transformieren. An jedes Datenmodell, welches über SAX geparst wurde, werden beim Einfügen in die Datenbank entsprechende, nur für die Applikation relevante Werte geknüpft, wie beispielsweise die Farbe der Punktobjekte, oder die aktuell ausgewählten Layer eines WMS-Dienstes. Da das WMS-Kartenmaterial ständig aktuell abgefragt wird, läuft die Kommunikation dort direkt vom WMS-Server (Kartenservice) zur *GoogleMaps-View* auf dem mobilen Gerät.
Die 3D-Objekte werden in einer selbst implementierten View dargestellt. Für die Repräsentation der 3D-Geoobjekte wird von der mobilen Applikation zur Zeit nur das GOCAD-ASCII-Format unterstützt. Dieses Dateiformat kann unmodifiziert auf dem Flash-Speicher des Gerätes vorgehalten werden.

3.1 Zugriff auf einen Web Map Service

Ein Web Map Service (WMS) [dLB02] erzeugt Karten aus georeferenzierten Daten. Dabei ist eine Karte nur eine Visualisierung der Daten als Rasterbild, nicht die Daten selbst. In der WMS-Spezifikation [dLB02] werden zwei Operationen zur Implementation vorgeschrieben, die beide über einen HTTP-Request angesprochen werden. Die *GetCapabilities*-Anfrage an einen WMS-Server liefert eine XML-Antwort mit allen Metadaten die dazu benutzt werden können, über eine *GetMap*-Anfrage Karten in verschiedenen Projektionen und Bildformaten für eine umschließende Bounding Box (BBOX) von dem WMS-Server zu erhalten.
Der WMS besteht aus sogenannten *Layern* in einer Baumstruktur. Ein WMS enthält genau eine solche Baumstruktur. Jeder *Layer* unterstützt die Projektionen seines Vaters und eventuell weitere. In der *GetMap*-Anfrage an den WMS-Server können benachbarte *Layer*, auch aus verschiedenen Ästen, abgefragt werden. Sie alle müssen dabei aber dieselbe Projektion unterstützen. Die Abfrage eines Vater-*Layers* macht die Abfrage der Söhne obsolet, denn beim Absteigen in die Baumstruktur wird davon ausgegangen, das die Söhne in ihrer geographischen Ausdehnung echte Teilmengen der Väter sind. Diese Ausdehnung wird neben vielen anderen Metadaten wie Datenquelle und diverse Darstellungsoptionen mit in der *GetCapabilities*-Antwort des WMS-Servers übergeben.
Zur Implementation eines mobilen WMS-Klienten reicht es zunächst, jeweils den *GetCapabilities*-Response des WMS-Servers zu parsen und die wichtigen Metadaten in eine SQLite-Datenbank zu schreiben. Aus der Spezifikation ergeben sich die folgenden Eigenschaften als wichtig: Die *Layer*-Baumstruktur mit den jeweils zulässigen Projektionen

und, soweit vorhanden, der Datenquelle und Legende, dazu Quelle und Abfrageadresse des Services. Um Material aus verschiedenen WMS-Layern übereinander darstellen zu können, werden nur solche Layer unterstützt, die im *GetMap*-Response transparente Karten im PNG-Format bereitstellen können. Die Benutzereinstellungen zu gerade dargestellten WMS-*Layern*, ausgewählten *Layern* und Projektionen werden direkt an das geparste Datenmodell gehängt. Die eigentliche Schwierigkeit besteht allerdings in einer effizienten Abfrage des Kartenmaterials.

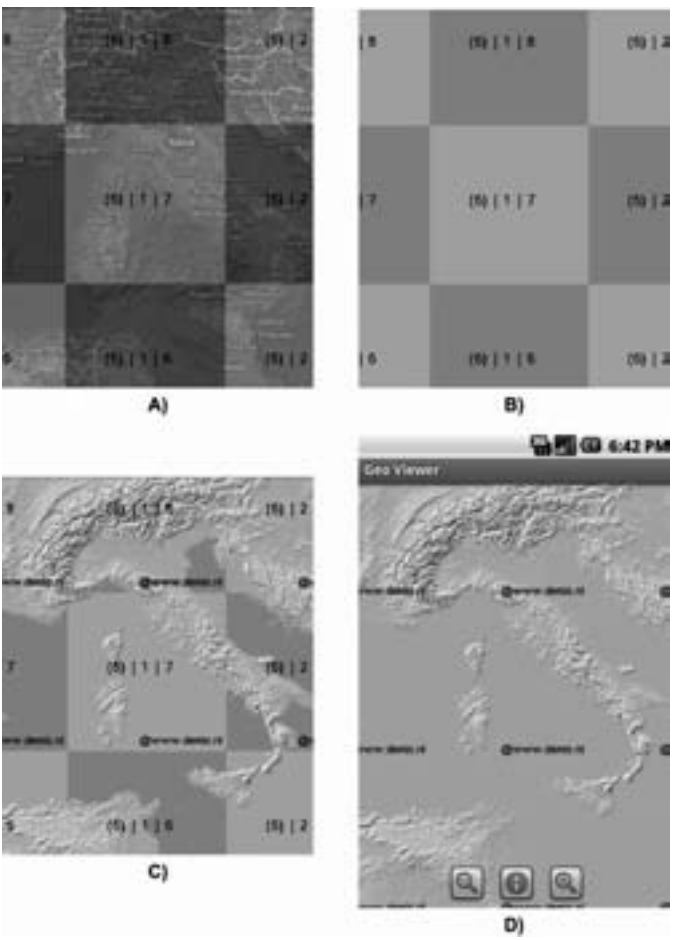

Abbildung 3: Tile Caching für WMS

Abbildung 3 verdeutlicht, wie zur Darstellung eines WMS-Layer vorgegangen wird. Dies geschieht mit sogenanntem *Tile Caching*. Das bedeutet, dass eine Karte nicht als ganzes in einem Request vom WMS-Server abgefragt wird, sondern gewissermaßen *zerstückelt* in *Tiles* (Kacheln). Diese werden dann, je nachdem welcher Teil des WMS-Layers sichtbar ist, wieder zusammengesetzt.

Der erste Schritt (Abb. 3 **A**) besteht darin, zu berechnen, welche Teile für den gerade sichtbaren Bereich abgefragt werden müssen. Um eine spätere Speicherung der Daten zu ermöglichen, wird eine *Kachel* eindeutig über die aktuelle Zoomstufe und einen Indikator identifiziert. Der Indikator berechnet sich aus der applikationsweit gleichen Größe der *Kacheln* und dem Abstand zum Ursprung des Geographischen Koordinatensystems. Da der Einfachheit halber auf die vorhandene GoogleMaps-API zurückgegriffen wird, sind die Koordinaten alle in einer abgewandelten Mercator Projektion mit EPSG-Code[eps10] EPSG:3857, womit der Ursprung auf dem Schnittpunkt des 0. Breiten- (Äquator) und 0. Längengrades (Greenwich) fällt.
Für ein und dieselbe Zoomstufe bleibt die Aufteilung in *Kacheln* somit gleich. In Schritt **B** ist diese eindeutig bestimmt, zusammen mit den Eckpunkten der *Tiles* in Geo- und Bildschirmkoordinaten. Im nächsten Schritt **C** wird für jede sichtbare *Kachel* die entsprechende Karte geladen, mit ihrem eindeutigen Schlüssel versehen und im Zwischenspeicher vorgehalten. Unter **D** ist dann das eigentliche Ergebnis sichtbar.
Wie erwähnt werden die Kacheln in einen Zwischenspeicher geladen und dort eine gewisse Zeit vorgehalten. Dies richtet sich danach, wie lange diese schon nicht mehr sichtbar waren und wie groß der Heap der Virtual Machine ist. Die Anzahl der Abfragen kann dadurch deutlich reduziert werden. Als zusätzliche Optimierung werden Kacheln im nahen Umfeld des sichtbaren Gesamtausschnittes nebenläufig geladen, um spätere horizontale und vertikale Bewegungen schneller zu machen.

3.2 Nutzung der Google Maps API

Im Gegensatz zur restlichen API ist die GoogleMaps-API nicht Freie Software sondern proprietär. Es ist jedoch schwierig Funktionen wie die Umrechnung von Geo- in Bildschirmkoordinaten effizient zu implementieren, deswegen wurde vorerst auf diese API gebaut. Die Einschränkungen liegen jedoch auf der Hand. Die Geokoordinaten werden in sechstelligen, ganzen Zahlen vorgehalten und die API unterstützt nur die sehr spezielle GoogleMaps-Projektion. Dabei handelt es sich, wie oben bereits genannt, um eine abgewandelte Mercator Projektion, so dass eine Karte beispielsweise im Vergleich zu WGS84 [Bar05, S. 215] zu den Polen hin gestreckt würde. Da kaum ein WMS-Server zur Zeit diese Projektion unterstützt, muss in den meisten Fällen mit einer gewissen Ungenauigkeit gerechnet werden.
Die Lösung für diese Problematik wäre die Implementation einer eigenen *Maps-API*, die unterschiedliche Projektionen und das *Tile Caching* direkt unterstützt. Vergleichbare Projekte sind *osmdroid* [Gra10] und *gvSIGDroid* [Rei09]. Ersteres unterstützt allerdings nur eine Projektion, letzteres hat das Desktop-GIS *gvSIG* [L$^+$10] als Grundlage und baut damit auf ein vollständiges GIS auf, welches für ein mobiles Endgerät zu umfangreich erscheint.

3.3 Zugriff auf einen Sensor Observation Service

Ein Sensor Observation Service (SOS) [NP07] dient zur Abfrage und Bereitstellung von Sensordaten. Dabei kann es sich um ortsgebundene oder bewegliche Sensoren handeln. Abbildung 4 verdeutlicht das von der Schnittstellenspezifikation vorgegebene Abfrageschema eines SOS-Services.

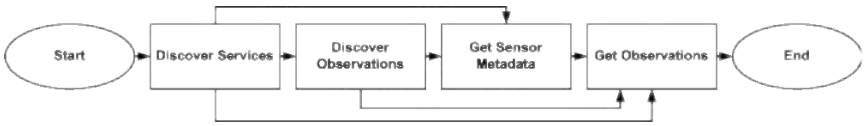

Abbildung 4: Schema zur Abfrage von Sensor Observation Services[NP07, S. 14, Fig 7.2]

Im ersten Schritt wird entschieden, welcher Service für die folgenden Abfragen genutzt werden soll. Dazu wird üblicherweise ein OGC-Catalogue Service (CS) verwendet, welcher es ermöglicht, für den Nutzer relevante Services bezüglich der betrachteten Positionen oder des benötigten Datensatzes zu finden. Ein SOS-Server verwaltet die Abfragen an unterschiedliche Sensoren und stellt verschiedene *Offerings* über eine *GetCapabilities*-Abfrage zur Verfügung. Die *Offerings* stellen jeweils eine Ansammlung von Sensoren mit ähnlichem Inhalt dar und sind vergleichbar mit einem WMS *Layer*. Verschiedene *Offerings* überdecken sich dabei aber nicht nur in ihrer geographischen Ausdehnung, sondern vor allem in den repräsentierten Daten. Zum Beispiel könnte eine Wetterstation in einem *Offering* ausschließlich zur Luftdruckmessung und in einem anderen *Offering* zur Bereitstellung vielfältiger meterologischer Daten genutzt werden. Die Entscheidung für ein *Offering* wird im Schritt *Disvover Observations* getroffen. An dieser Stelle stehen dem Nutzer also Metainformationen über den verwendeten Service und das Offering, wie etwa die Quelle der Daten oder Serviceinformationen, zur Verfügung. Er kann im folgenden Schritt (*Get Sensor Metadata*) vom SOS-Server Metainformationen über einen einzelnen Sensor mit einer *DescribeSensor*-Abfrage erhalten, wie beispielsweise die von dem Sensor bereitgestellte Methode der Datenabfrage (*Procedure*), sowie die geographische Position oder Bauart des Senors. Die eigentlichen Datensätze können vom SOS-Server schließlich im Schritt *Get Observations* mit einem *GetObservation*-Request abgefragt werden. Die Anfrage kann sich dabei direkt an einen Sensor richten oder auf einem höheren Abstraktionsgrad über ein *Offering* erfolgen.
Ein SOS-Service wird über HTTP-Post mit XML-Dateien angesprochen und liefert das Ergebnis wiederum in einem XML-Format an den Klienten zurück. Für jede der oben aufgeführten Operationen gibt es wiederum eine Vielzahl zu beachtender Schemata, die die Abfragen aufwändig gestalten. Insgesamt verursacht der vorgeschriebene Weg im Kontext mobiler Applikationsentwicklung also einen viel zu hohen Rechen- und Speicheraufwand. Bei der Implementation des mobiles SOS-Klienten wird darum zunächst gänzlich auf einen Catalogue Service (CS) verzichtet und der Schritt *Discover Observations* fallengelassen. Es wird davon ausgegangen, dass der Nutzer bereits entschieden hat, welchen Service er verwenden will. Unabdingbar ist hingegen *Discover Observations*, da in einem *GetCapabilities*-Response des SOS-Servers eine Liste der möglichen Abfrageparameter

für die *GetObservation*-Requests enthalten ist. In einem *DescribeSensor*-Request kann keine spezielle Prokjektion für die gelieferten Geodaten gefordert werden und der Großteil der Informationen wird zusätzlich im *GetObservation*-Response geliefert. Deswegen wird der Schritt *Get Sensor Metadata* ebenfalls nicht von der mobilen Applikation unterstützt.

In der Implementation der mobilen Applikation werden nach jedem Request an einen SOS-Server die vom Server zurückgelieferten XML-Antworten mittels SAX geparst und die gewonnenen Daten in eine SQLite Datenbank übertragen. Die vielfältigen und besonders rechenintensiven String-Operationen, die vor allem zum Parsen eines *GetObservation* Response nötig sind, konnten dabei nicht umgangen werden, da sie von der SOS-Spezifikation quasi vorausgesetzt werden. Das größte Problem ist jedoch, das ein *GetCapabilities* Response des SOS-Servers zwar eine Liste der *Offerings* und der in diesen enthaltenen Messstellen (*Features*) und beobachteten Phänomene (*Poperties*) enthält, nicht jedoch die zugehörigen geographischen Koordinaten. Diese werden erst mit einem entsprechenden *GetObservation* Response geliefert. Damit können relevante Sensoren nicht etwa wie bei den Karten in einem WMS in einer BBOX ermittelt werden, sondern es ist zunächst notwendig, alle enthaltenen *Features* bereits vorher einmal abgefragt zu haben, um eine Liste von Punktdaten bereitstellen zu können. Abbildung 5 zeigt schließlich die Visualisierung die über den Pegelonline [peg10] SOS bereitgestellten Wasserstandsdaten eines Sensors vor Norderney.

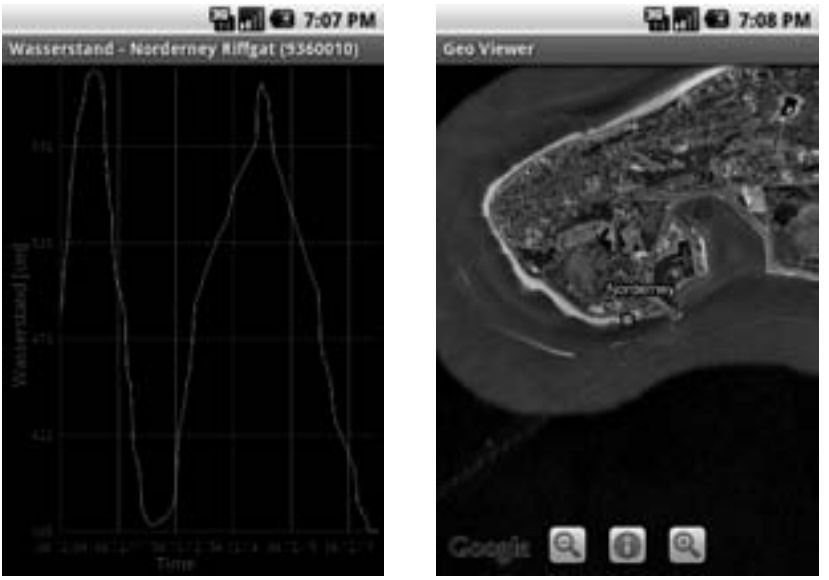

Abbildung 5: Verlauf von abgefragten Sensordaten (*links*), Visualisierung von Punktdaten (*rechts*)

3.4 Visualisierung von 3D Geodaten

Bisher wurden nur zweidimensionale Geodaten behandelt, doch zur genaueren Beurteilung von geographischen Informationen ist es oft hilfreich diese realitätsnäher, also dreidimensional, darzustellen. Die Visualisierung von Datensätzen im GOCAD-ASCII-Format [Pro10] ist hierzu ein erster Ansatz und bietet die Möglichkeit abzuschätzen, wie leistungsstark heutige mobile Endgeräte für die 3D-Visualisierung sind.
In Android geschieht 3D-Visualisierung mit Hilfe von Open GL ES 1.0. Open GL ES besitzt zwar gewisse Einschränkungen in der Funktionalität gegenüber OpenGL, aber die nötigen Features zur Darstellung von Vektordaten mit einfachen Beleuchtungseffekten werden bereitgestellt. Die Schwierigkeit besteht im effizienten Überführen des GOCAD-Datensatzes in die von OpenGL ES akzeptierte Buffer-Form. Bei der Visualisierung wurde zudem darauf geachtet, dass die letztendlich dargestellten Objekte bzgl. Größe, Anzahl, Ausdehnung im Koordinatensystem und deren Ursprungskoordinaten völlig unterschiedlich sein können. Deswegen lag ein besonderes Augenmerk auf der dynamischen Anpassung der Parameter für Zoom- und Walkfunktionen.
Zu große Datensätze kann ein mobiles Endgerät schlichtweg nicht effizient darstellen und eine vorgehende Generalisierung der verwendeten Daten, oder sogar die Verwendung einer stufenweisen, von der Größe des sichtbaren Ausschnitts abhängigen Generalisierung, ist ebenfalls zu rechenintensiv. Deswegen muss darauf geachtet werden, das die darzustellenden Datensätze in ihrem Umfang eine durch die Hardware des Gerätes gegebene Oberschranke nicht überschreiten. Dies kann natürlich einerseits „manuell“ durch den Benutzer erfolgen, oder aber durch den Server von dem die Daten abgefragt werden. Hierfür soll in Zukunft eine Schnittstelle für die 3D/4D-Geodatenbank DB4GeO [B$^+$09] implementiert werden. Abbildung 6 zeigt die für die mobile Applikation implementiert View zur Darstellung von geologischen 3D-Daten mit Hilfe von Open GL ES.

3.5 GeoRSS als Ansatz für einen Frühwarnsystemclienten

Geo-enabling RSS Feeds (GeoRSS)-Feeds [Ree06] [geo10] dienen zur Bereitstellung von raum- und ereignisbezogenen Daten. Diese erweitern die weit verbreitete Real Simply Syndication (RSS) [C$^+$10] Technologie mit einer Georeferenzierung über eingebundene GML- oder W3CGeo- [w3c10] *Tags*. Da sich GeoRSS direkt bekannter Standards bedient, ist die Implementation eines GeoRSS-Readers verglichen mit den oben vorgestellten Standards relativ einfach. Die abgefragten Daten werden wiederum mit einem SAX-Parser verarbeitet und die gewonnen Daten werden in einer SQLite Datenbank für die offline-Nutzung auf dem mobilen Gerät vorgehalten.
Der Anwendungsfall gestaltet sich spannender. Im Kontext einer mobilen Nutzung können GeoRSS-Feeds es ermöglichen, den Endnutzer, nach einer Abonnierung des Dienstes, in von ihm festgelegten regelmäßigen Abständen, über wichtige Ereignisse mit Raumbezug wie zum beispielsweise Naturkatastrophen zu informieren. Somit wäre es möglich, die vorgestellte Applikation als Klient für ein Frühwarnsystem einzusetzen. Weiterhin würde die Erweiterung der bisherigen Anwendungen von RSS als Mittel der Nachrich-

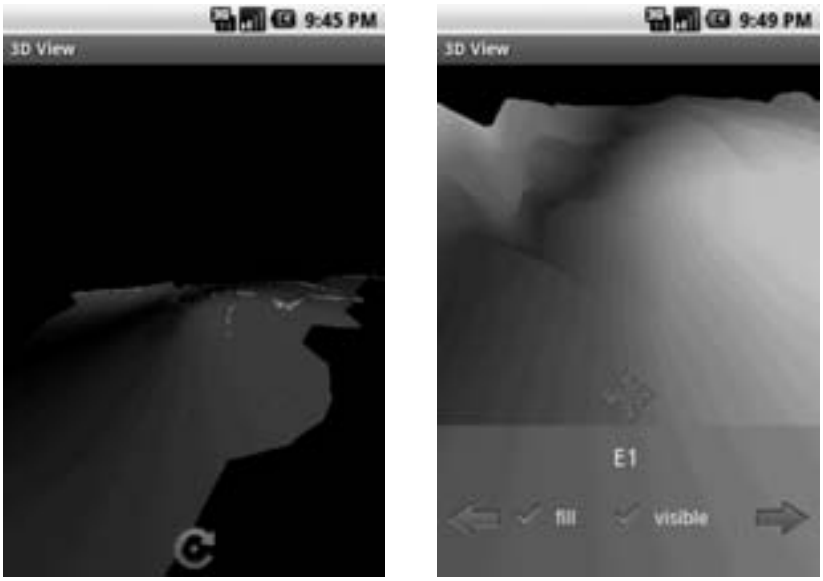

Abbildung 6: 3D - Visualisierung von Oberflächendaten vom „Winkelgrat“ bei Balingen in der schwäbischen Alb

tenübermittlung zu GeoRSS es ermöglichen, Nachrichten standortbezogen abzufragen und damit im Umkehrschluss Nachrichten für einen geographischen Interessensraum bereitzustellen. Naturereignisse könnten beispielsweise direkt für diejenigen Nutzer gefiltert werden, die sich in der Nähe des Ereignisses befinden. Die eigentliche Herausforderung besteht damit in der regelmäßigen und effizienten Synchronisation der abonnierten GeoRSS Feeds.
In Kombination mit den oben vorgestellten Features der mobilen Applikation *GeoViewer* ist es dem Nutzer mit der Benachrichtigung über ein Ereignis möglich, weitere relevante Umgebungsdaten abzufragen. Dem professionellen GIS-Anwender, wie etwa Geologen oder Raumplanern, ermöglicht es damit, vor Ort bereits eine erste Analyse der Situation durchzuführen.

4 Zusammenfassung und Ausblick

Mit der vorgestellten mobilen Applikation *GeoViewer* wurde die Eignung verschiedener Ansätze zur mobilen Visualisierung von raumbezogenen zwei- und dreidimensionalen Daten untersucht. Bei der Visualisierung von Kartenmaterial mit Hilfe eines Web Map Service (WMS) zeigte sich, dass die verwendete GoogleMaps-API für die weitere Entwicklung eines mobilen Geoinformationssystems ungeeignet ist. Diese soll in Zukunft durch eine eigene *Maps-API* ersetzt werden. Einschränkungen bei der Darstellung und Ab-

frage von Sensordaten über einen Sensor Observation Service (SOS) ergaben sich durch die für mobile Geräte zu umfangreiche Servicespezifikation selbst. Wogegen für die Darstellung von dreidimensionalen Daten mit OpenGL ES noch ein geeigneter Webservice zur Datenabfrage angebunden werden muss. Insgesamt bestand die größte Herausforderung bei der Implementation immer in der Reduktion der verwendeten Standards und Formate auf die relevanten und geeigneten Funktionen unter Berücksichtigung der immer noch knappen Hardwareresourcen mobiler Endgeräte.
Die Verwendung von GeoRSS in der mobilen Applikation zur Darstellung ereignisbezogener, georeferenzierter Daten zeigte die mögliche Eignung eines mobilen Gerätes als Träger eines Frühwarnsystem-Klienten. In Verbindung mit den anderen vorgestellten Funktionen könnte direkt nach der Benachrichtigung durch GeoRSS eine erste raumbezogene Analyse des Ereignisses vorgenommen werden. Eine Einschränkung von GeoRSS besteht aber darin, das die mobile Applikation für den Erhalt der Nachricht selbst Sorge tragen muss und nicht automatisiert benachrichtigt wird, sobald ein Ereignis eintritt. Als Alternative könnte die Anbindung an einen Sensor Alert Service (SAS) [Sim07] dienen, welcher die automatisierte Benachrichtigung eingetragener Klienten unterstützt, im Gegensatz zu GeoRSS jedoch auf die Verarbeitung sensorbezogener Nachrichten spezialisiert ist.

Literatur

[and10a] Android, offizielle Entwicklerseite. `http://developer.android.com`, 2010. [Online; Stand 24.01.2010].

[and10b] What is Android. `http://developer.android.com/guide/basics/what-is-android.html`, 2010. [Online; Stand 24.01.2010].

[B+07] Martin Breunig et al. Development of suitable Informationsystems for early warning systems. In *Geotechnologien Science Report No. 10*, Seiten 113–123, 2007.

[B+09] Martin Breunig et al. DB4GeO: Developing 3D geo-database services. In Philippe De Maeyer, Tijs Neutens, Marijke De Ryck, Hrsg., *3D GeoInfo 2009*, Seiten 45–51, 2009. Proceedings of the 4th International Workshop on 3D Geo-Information.

[Bar05] Norbert Bartelme. *Geoinformatik, Modelle Strukturen Funktionen*. Springer-Verlag Berlin Heidleberg, 2005.

[C+10] Rogers Cadenhead et al. RSS 2.0 Specification. `http://www.rssboard.org/rss-2-0`, 2010. [Online; Stand 24.01.2010].

[Com10] Maemo Community. Maemo, offizielle Entwicklerseite. `http://maemo.org/development/`, 2010. [Online; Stand 24.01.2010].

[dLB02] Jeff de Lay Beaujardièr. *Web Map Service Implementation Specification*. Open GIS Consortium Inc., 1.1.1. Auflage, 1 2002. Reference Number OGC 01-068r3.

[ecl10] Eclipse Project. `http://www.eclipse.org/`, 2010. [Online; Stand 24.01.2010].

[eps10] EPSG Homepage. `http://www.epsg.org`, 2010. [Online; Stand 24.01.2010].

[Fou10] Symbian Foundation. Symbian OS, offizielle Seite. `http://www.symbian.org/`, 2010. [Online; Stand 24.01.2010].

[geo10] GeoRSS. http://www.georss.org, 2010. [Online; Stand 24.01.2010].

[Gra10] Nicolas Gramlich. OSMDroid. http://code.google.com/p/osmdroid/, 2010. [Online; Stand 24.01.2010].

[Inc10] Apple Computer Inc. iPhone OS, offizielle Entwicklerseite. http://developer.apple.com/iphone/, 2010. [Online; Stand 24.01.2010].

[L+10] Mario Flores Lanuza et al. GvSIG Homepage. http://www.gvsig.gva.es/eng/inicio-gvsig/, 2010. [Online; Stand 24.01.2010].

[Men10] Mathias Menninghaus. Quellcode der Applikation GeoViewer. http://code.google.com/p/mobiledroidgis/, 2010. [Online; Stand 24.01.2010].

[Mic10] Microsoft. Windows Mobile, offizielle Entwicklerseite. http://developer.windowsphone.com/Default.aspx, 2010. [Online; Stand 24.01.2010].

[MS04] Bela Mutschler und Günther Specht. *Mobile Datenbanksysteme*. Springer Verlag Heidelberg Berlin, 2004.

[NP07] Arthur Na und Mark Priest. *Sensor Observation Service*. Open GIS Consortium Inc., 1.0.0. Auflage, 10 2007. Reference Number OGC 06-009r6.

[ogc10] Open Geospatial Consortium Homepage. http://www.opengeospatial.org/, 2010. [Online; Stand 24.01.2010].

[ope10] Open Handset Alliance. http://www.openhandsetalliance.org, 2010. [Online; Stand 24.01.2010].

[Pal10] Palm. Palm WebOS, offizielle Entwicklerseite. http://developer.palm.com/, 2010. [Online; Stand 24.01.2010].

[peg10] Pegelonline Webservices. http://www.pegelonline.wsv.de, 2010. [Online, Stand 24.01.2010].

[Pro10] GOCAD Project. GOCAD Project. http://www.gocad.org, 2010. [Online; Stand 24.01.2010].

[Ree06] Carl Reed. *An Introduction to GeoRSS: A Standard Based Approach for Geo-enabling RSS feeds.* Open GIS Consortium Inc., 1.0.0. Auflage, 7 2006. Reference Number OGC 06-050r3.

[Rei09] Christian Martin Reinhold. gvSIGDroid An open source GIS for the Android platform. Diplomarbeit, Universitat Jaume I, 2009.

[sax10] SAX Project. http://www.saxproject.org, 2010. [Online; Stand 24.01.2010].

[Sim07] Ingor Simonis. *OGC Sensor Alert Service Implementation Specification*. Open Geospatial Consortium Inc., 0.9.0. Auflage, 5 2007. Reference Number OGC 06-028r5, in Revision.

[sql10] SQLite. http://www.sqlite.org, 2010. [Online; Stand 24.01.2010].

[w3c10] W3C Geo Specification. http://www.w3.org/2003/01/geo/, 2010. [Online; Stand 24.01.2010].