FAKULTÄT FÜR !NFORMATIK

Programmierung von Multimediasensoren für mobile Informationsszenarien

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Medieninformatik

eingereicht von

Johannes Spreitzer
Matrikelnummer 0307362

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer: Ao. Univ. Prof. Dr. Horst Eidenberger

Wien, 19.1.2010

(Unterschrift Verfasser) (Unterschrift Betreuer)

Erklärung

Johannes Spreitzer
Tanngraben 4
A-3363 Winklarn

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.“

Wien, am 19.1.2010

_____

Unterschrift

Kurzfassung

Die vorliegende Arbeit beschäftigt sich mit der Software-Entwicklung und Medienverarbeitung auf mobilen Endgeräten unter dem Betriebssystem Android. Im Zentrum steht ein praktisches Projekt, das die Erstellung eines Programms als Ziel hat, das kontinuierlich die aktuelle Position des Geräts ermittelt, speichert und bei Bedarf wieder abrufen kann. Das Projekt soll auch einen Schrittzähler implementieren und das Erstellen von Panoramafotos zulassen. Zusätzlich soll es die gewonnenen Daten in geeigneter Weise visualisieren.

Zu Beginn der Arbeit werden das Android-System mit seinen Basiskonzepten vorgestellt und die zusätzlichen im Projekt verwendeten Technologien erarbeitet. Bei diesen Technologien handelt es sich um GPS-Tracking, Akzelerometer-Sensoren, Panorama-Stitching und Visualisierungsmethoden im Einsatz am mobilen Endgerät sowie im Internet. Anschließend wird das praktische Projekt mittels UML-Diagrammen modelliert, implementiert und die gewonnenen Ergebnisse werden erläutert.

Abstract

This thesis deals with software development and media processing on mobile handsets running the Android operating system. It is centered round a practical project, which aims to develop a program that continually tracks and saves the device's location. Furthermore the program should implement a stepcounter and make it possible to create panoramaphotos. Additionally it should display all the collected data in an appropriate manner.

At the outset the Android system with its basic concepts is introduced and the additional technologies drawn upon in the project are elaborated. These technologies include GPS-tracking, accelerometer-sensors, panorama-stitching and visualization-techniques as they come into operation on the mobile handset as well as on the web. Finally the practical project is modeled using graphical UML-diagrams and implemented, with the results achieved being discussed.

Danksagung

Ich bedanke mich bei allen Menschen, die mir während der Erstellung dieser Arbeit mit persönlichem oder fachlichem Rat zur Seite standen, sowie bei der Fakultät für Informatik für die finanzielle Unterstützung.
Besonderer Dank gilt Herrn Ao. Univ. Prof. Dr. Horst Eidenberger für die Betreuung dieser Diplomarbeit.

Konventionen

Innerhalb der vorliegenden Arbeit habe ich folgende Übereinkünfte bezüglich Satz- und Schreibstil getroffen:

- Um den Lesefluss nicht unnötig zu behindern, habe ich Quellenangaben als Fußnoten ausgeführt.

- *Kursiver Text* zeichnet Eigennamen, Fachausdrücke und betont wichtige Eigenschaften aus. Ist der Absatz auch eingerückt und unter Anführungszeichen gesetzt, handelt es sich um ein wörtlich aus der Literatur übernommenes Zitat.

- `Mit nichtproportionaler Schrift` werden Quellcodes, Systembezeichnungen, Internetadressen, etc. dargestellt.

- Auf geschlechtsneutrale Formulierungen wird zur leichteren Lesbarkeit verzichtet. Derartige Begriffe meinen aber stets beide Geschlechter.

Abkürzungsverzeichnis

A2DP	Advanced Audio Distribution Profile
A-GPS	Assisted GPS
ADB	Android Debug Bridge
ADT	Android Development Tools
ANR	Application Not Responsive
API	Application Programming Interface
Dalvik VM	Dalvik Virtual Machine
DDMS	Dalvik Debug Monitor Service
DBMS	Database Management System
DPI	Dots per Inch
ER	Entity Relationship
GPRS	General Packet Radio Service
GPS	Global Positioning System
GUI	Graphical User Interface
IDE	Integrated Development Environment
IMS	IP Multimedia Subsystem
IP	Internet Protocol
IPC	Inter Process Communication
IPTV	IP Television
J2ME	Java Platform 2, Micro Edition
JDK	Java Development Kit
LBS	Location Based Service
MSI	Mobile Spatial Interaction
NDK	Native Development Kit
OHA	Open Handset Alliance
QoS	Quality of Service
RIM	Research In Motion
SCO	Synchronous connection oriented [link]
SDK	Software Development Kit

SMS	Short Message Service
UI	User Interface
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
VM	Virtual Machine
VoIP	Voice over IP
WLan	Wireless Local Area Network
WMS	Web Map Service

Inhaltsverzeichnis

Kapitel 1

Einleitung

Diese Diplomarbeit mit dem Titel *Programmierung von Multimediasensoren für mobile Informationsszenarien* beschäftigt sich mit der Software-Entwicklung und Medienverarbeitung unter *Android*, einem Betriebssystem für mobile Endgeräte. Sie führt in die grundlegenden Konzepte von Android ein und erklärt die technischen Hintergründe eines konkreten Projekts, das in weiterer Folge genau ausgearbeitet, modelliert und in Form eines Prototyps praktisch umgesetzt wird. Anschließend werden die aus dem Projekt gewonnenen Erkenntnisse und Ergebnisse präsentiert sowie eine abschließende Zusammenfassung gegeben.

Die Grundzüge des praktischen Projekts entstanden bei einer Diskussion über die Sicherheit des Skitourengehens. Angesichts der dabei drohenden Gefahren von Lawinenabgängen, Abstürzen u.ä. und den Schwierigkeiten, Verunglückte zu finden, erschien mir der Bedarf an einem Hilfsmittel zur Ortung und zur automatischen Alarmierung gegeben. Die technischen Voraussetzungen für ein derartiges Hilfsmittel bietet ein modernes Mobiltelefon, das GPS-Sensoren zur Positionsbestimmung sowie verschiedene Kommunikationskanäle zum Herbeirufen von Hilfe mit sich führt. Dieser Anwendungsfall bildet mit einigen Funktionserweiterungen die Basis für das hierin erarbeitete praktische Projekt.

1.1 Projektziel

Das Projektziel definiert sich damit wie folgt: Erstellung eines Programms, das die sich verändernden Standorte des Geräts ermittelt und aufzeichnet. Mithilfe des im Mobiltelefon integrierten Akzelerometers soll ein Schrittzähler implementiert werden, der Aufschluss über die aktuellen Bewegungsmuster gibt und damit als

Basis zur Detektion einer möglichen Verunglückung dient.

Eine Verunglückung wird dann angenommen, wenn der Schrittzähler eine gewisse Zeit keine Schritte eruiert. Mit dem Eintreten dieses Ereignisses werden sodann die Rettungsroutinen aktiviert und auf allen möglichen Kanälen (Mobilnetz, Bluetooth und WLan) Notfallmeldungen inklusive genauer Ortsangabe abgesetzt. Da gerade im Anwendungsfall des Tourengehens nicht von einer Mobilfunk-Versorgung ausgegangen werden kann, ist die Methodik der persönlichen Funknetze (Bluetooth und WLan) besonders wertvoll. Vor allem unter dem Gesichtspunkt, dass Skitouren meist in einer Gruppe unternommen werden, ist es sinnvoll, sich im Verunglückungsfall mit den anderen Teilnehmern zusammenzuschließen. Mit Glück sind nicht alle Teilnehmer involviert, womit diese die Verunglückten anhand der mitgesendeten genauen Ortsangabe retten können. Weiters soll es auch die Möglichkeit geben, dass jemand anders den Standort anfordert und das Gerät den aktuellen Standort selbsttätig zurücksendet. Damit kann man beispielsweise verschollen geglaubte Mitglieder seiner Gruppe ausfindig machen.

Weil gerade im alpinen Gelände Panoramafotos sehr beliebt sind, widme ich einen Teil des Projekts auch dem Erstellen solcher Panoramas. Die Anforderung ist dabei, die Panoramas mittels Schwenken der Kamera über die Landschaft automatisch zu erstellen. Zusätzlich sollen die gewonnenen Daten (Routen, Schrittanalysen, Panoramas, etc.) in geeigneter Form am Gerät wie auch im Internet visualisiert werden können.

Für die genaue Anforderungsanalyse und Modellierung sei auf Kapitel 3 verwiesen. Angemerkt sei, dass dieses Programm auch für andere Szenarien als dem Skitourengehen genutzt werden kann. So kann man es generell für Outdoor-Aktivitäten wie Radfahren, Laufen, Wandern, Skifahren, Klettern, etc. nutzen, wofür Analysen bezüglich Distanzen, Geschwindigkeiten und Höhenmeter interessant erscheinen. Gleichzeitig ist in diesen Anwendungsfällen auch das Standort-Anfrage-System sinnvoll, um entweder die Kollaboration innerhalb einer Gruppe zu vereinfachen (Fragestellung *Wo ist Person ...?*) oder ermächtigten Personen die Möglichkeit zu geben, seinen aktuellen Standort anzufragen.

Ein weiteres Szenario bildet das Sightseeing. Dabei kann das Programm entweder als Orientierungshilfe dienen (Fragestellungen *Wo bin ich?* und *Wo ist ...?*) bzw. im Nachhinein Aufschluss über die getätigte Route geben. Weiters können von Sehenswürdigkeiten, Plätzen, Aussichtsplattformen, etc. Panoramafotos erstellt werden, welche georeferenziert in der aufgezeichneten Route in der Karte erscheinen. Ist man zu zweit oder mehrt unterwegs, bietet auch hier das Standort-Anfrage-System eine einfache Möglichkeit, sich wieder zu finden wenn man sich verloren hat. Interessante Anwendungsfälle des Programms ergeben sich auch beim Auto-

fahren oder zum Aufspüren des Geräts, nachdem es gestohlen wurde oder man es verlegt hat.

Kapitel 2

Hintergrund

Dieses Kapitel beleuchtet die dem praktischen Projekt zugrunde liegenden Technologien. Neben dem Betriebssystem *Android* als Systembasis werden auch die Technologien bezüglich *Tracking und Location Based Services*, *Akzelerometer*, *Kommunikation*, *Panorama-Stitching* und *Visualisierung* thematisiert.

2.1 Android-Betriebssystem

2.1.1 Einführung und Geschichte

Abbildung 2.1: Logo und Schriftzug der OHA. [15]

Am 5. November 2007 wurde von der Firma Google und weiteren 33 Unternehmen die Gründung der *Open Handset Alliance* (OHA) angekündigt und gleichzeitig die

Abbildung 2.2: Logo von Android. [16]

erste Version des *Android Software Development Kit* (SDK) veröffentlicht.[1] Die offiziellen Erkennungszeichen der OHA und von Android sind in Abbildung 2.1 sowie Abbildung 2.2 ersichtlich.

Bereits am 9. Dezember 2008, zirka ein Jahr nach ihrer Gründung, traten der OHA 14 weitere Mitglieder bei. Damit gehörten ab diesem Zeitpunkt sieben Netzbetreiber, neun Halbleiterfirmen, vier Endgerätehersteller, zehn Softwarefirmen und vier Vermarktungsfirmen der OHA an. Sie entwickeln gemeinsam die Plattform *Android*.[2] Eric Schmidt, CEO von Google, kommentierte dazu am OHA-Gründungstag:

> *„Diese Partnerschaft wird helfen, das Potenzial der mobilen Technologien für Milliarden von Nutzern weltweit zu realisieren. [...] Unsere Vision ist es, dass die starke Android Plattform, die wir heute enthüllen, Tausende von unterschiedlichen Mobilfunkgeräten unterstützt“*[3]

Mit der mobilen Nutzung des Internets erklärt sich auch die große Motivation von Google, das Androidprojekt voranzutreiben: Googles wirtschaftliches Standbein stellt die Online-Werbung dar und „mit diesen Geräten wird die Online-Werbung gezielter werden, da diese Mobilfunkgeräte persönliche Geräte sind. [...] Und dies bedeutet, dass der Wert der Werbung steigt. Der nächste große Trend in der Online-Werbung ist das mobile Internet“[4]

Damit tritt die OHA mit der federführenden Kraft von Google in direkte Konkurrenz zu Nokia und der *Symbian Foundation* mit derem etablierten Mobiltelefon-Betriebssystem *Symbian*. Die Reaktion Nokias blieb nicht aus und die Firma stellte kurz darauf ihre Strategie um. So ist geplant, die aktuell existierenden, verschiedenen auf Symbian basierenden Betriebssysteme zu vereinheitlichen und das

[1] Vgl. [5] S. 3
[2] Vgl. [4] S. 10
[3] [4] S. 14
[4] Vgl. [4] S. 15

System abschließend ebenfalls - wie Android - unter einer Open-Source-Lizenz zu veröffentlichen.

Einige der zusammengeschlossenen Unternehmen setzen jedoch auf beide Plattformen, also Android *und* Symbian. Zu diesen Firmen zählen u.a. die Namen Ericsson, LG, Motorola, Sony Ericsson, Samsung, T-Mobile, Texas Instruments und Vodafone.[1]

2.1.2 Eigenschaften von Android

Android ist ein Betriebssystem für mobile Geräte, vornehmlich für Mobiltelefone und wird auf unterster Ebene von einem Linux-Kernel angetrieben. Es bietet neben zahlreichen Bibliotheken und APIs zum geschützten Zugriff auf die Hardware eine Laufzeitumgebung sowie einige bereits vorinstallierte Basisapplikationen.[2] Neben der reinen Spezifikation zur Softwareplattform existiert aber auch „eine Hardware-Referenz, welche die Voraussetzungen eines mobilen Geräts spezifiziert, das Softwaresystem [darauf] zu betreiben.“[3] Des Weiteren ist das komplette Android-System seit dem 21. Oktober 2008 als Open-Source-Software verfügbar.[4] Damit fallen weder den Benutzern bzw. Drittentwicklern, noch den Hardwareherstellern etwaige Lizenzkosten an. Daher macht es vor allem dieser Aspekt für Endgerätehersteller lukrativ, in die Entwicklung von Android-Endgeräten zu investieren.[5]

Durch die Offenlegung der Quellcodes hat sich auch schnell eine Community gebildet, welche sich die daraus ergebenden Vorzüge zu Nutze macht: So erstellen bzw. erweitern manche Benutzer selbst ihre eigenen Android-Systeme oder portieren sie auf gängige Endgeräte, die eigentlich nicht für Android gedacht waren.[6] Anzumerken ist jedoch, dass die verschiedenen *Google Apps* (Maps, YouTube, Gmail, u.s.f.) nicht Teil der Android-Plattform sind und somit auch nicht open-source zur Verfügung stehen.[7]

Obwohl es möglich ist, C/C++ Softwareteile mittels des *Android Nativ Development Kit* (NDK) einzubinden und auszuführen, können Anwendungen ausschließlich für die Laufzeitumgebung Dalvik VM geschrieben und darin ausgeführt wer-

[1] Vgl. [4] S. 15f
[2] Vgl. [4] S. 1
[3] Vgl. [3] S. 4
[4] Vgl. [17]
[5] Vgl. [4] S. 1
[6] Vgl. [19]
[7] Vgl. [20]

den.[1] Zur Programmierung bedient man sich dem Konzept und der Sprache von Java, was durch seine große Verbreitung und Akzeptanz unter Developern den Einstieg in die Android-Entwicklung erleichtern und wiederum das Vordringen der ganzen Plattform selbst begünstigen soll.[2] So werden viele APIs der Java SE unterstützt und ein *Garbage Collector* übernimmt auch unter Android die automatische Speicherfreigabe.[3]

Jede Anwendung wird jeweils in einer eigenen Instanz der Dalvik VM, einer so genannten *Sandbox* (siehe Abschnitt 2.1.3), ausgeführt, wodurch eine Anwendung von allen anderen komplett abgeschottet wird. Auch auf das System kann eine Anwendung nur mithilfe der verfügbaren APIs zugreifen und das auch nur dann, wenn ihr zuvor die erforderlichen Rechte erteilt wurden (siehe Abschnitt 2.1.5). Das kommt vor allem der Robustheit und der Sicherheit des Systems zugute.[4] Damit Anwendungen trotz der Begrenzungen durch die Sandbox miteinander kommunizieren können (Interprocess Communication - IPC) wurden die in Android sehr grundlegenden Konzepte der *Broadcast Intents* sowie *Broadcast Receiver* entwickelt (siehe Abschnitt 2.1.5).

Sehr bemerkenswert ist weiters, dass unter Android *alle* Komponenten von Vornherein gleichberechtigt sind. So werden Anwendungen von Drittanbietern (bzw. auch nur einzelne Module) gleich behandelt wie die nativ vorinstallierten bzw. ist es möglich, „native Anwendungen durch andere zu ersetzen“.[5]

Da Android für mobile Geräte konzipiert wurde, ergeben sich durch deren technische und physikalische Eigenheiten einige beachtenswerte Aspekte. Dies betrifft vor allem die Hardwareressourcen, wie geringere CPU-Leistung und Speichergröße, die kleinere Bildschirmgröße, die Art der Bedienung sowie die Energieversorgung, verglichen mit herkömmlichen Desktoprechnern. Aber auch die per se standortungebundene Nutzung des Geräts wirft neue Bereiche auf (z.B. instabile Netzwerkanbindung), die beim Entwickeln einer Anwendung entsprechend mitberücksichtigt werden müssen.[6]

Zum besseren Verständnis wird noch angemerkt, dass Android keine *Java Micro Edition* (J2ME) - Implementation ist, Applikationen aber in der Programmiersprache Java geschrieben werden. Genausowenig ist Android ein *konkretes* Mobiltelefon, es spezifiziert jedoch sehr wohl gewisse Hardwareanforderungen. Es ist auch

[1] Vgl. [18]
[2] Vgl. [3] S. 10
[3] Vgl. [4] S. 2f
[4] Vgl. [4] S. 3
[5] Vgl. [3] S. 11
[6] Vgl. [4] S. 3

nicht „Googles Antwort auf Apples iPhone“. Während das iPhone eine „proprietäre Plattform einer einzigen Firma ist“, ist Android eine „offene Softwareplattform, die von der OHA dahingehend entwickelt wurde, dass sie auf jedem Gerät laufen kann, das die Anforderungen erfüllt.“[1]

2.1.3 Systemarchitektur

Die Beschreibung der Systemarchitektur, oftmals auch als *System Stack* oder *Software Stack* bezeichnet, ist das Thema dieser Sektion. Hier werden die Zusammenhänge der einzelnen Komponenten und Schichten (Layer) im Gesamten erläutert.

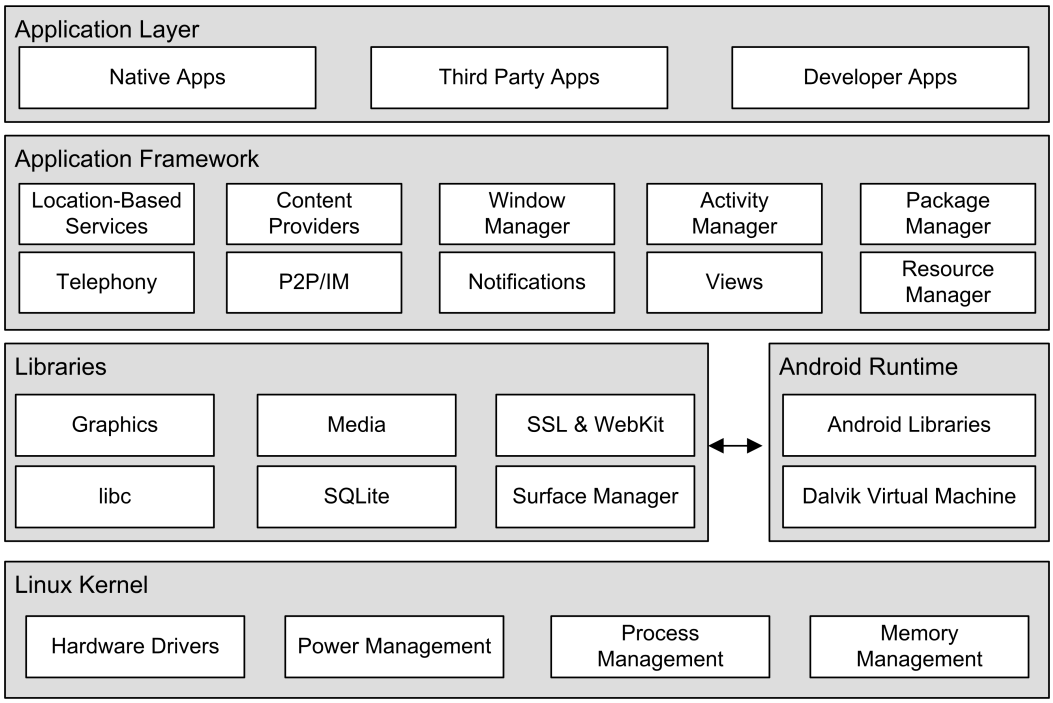

Abbildung 2.3: System Stack von Android. [3] S. 13

Abbildung 2.3 zeigt den System Stack innerhalb von Android. Seinee Schichten (beginnend mit der untersten) sind: *Linux Kernel*, *Libraries*, *Android Runtime*, *Application Framework* und *Applications*. Dabei steht jeder übergeordneten Ebene die gesamte Funktionalität der unteren Ebenen zur Verfügung.

[1] Vgl. [3] S. 3

Ein Linux-Kernel bildet die unterste und somit hardwarenächste Schicht. Er ist für das Hardware-, Energie-, Prozess-, Speicher- sowie Sicherheitsmanagement zuständig.

Die nächste Ebene bilden die C/C++ Standardbibliotheken (*Libraries*), die mittels einer Schnittstelle von Java aus ansprechbar sind und eine Menge an Basisfunktionalitäten bereitstellen. So bietet u.a. die *SQLite*-Bibliothek ein gängiges, in embedded Systems oft anzutreffendes *Datenbank Management System* (DBMS) an bzw. bedient man sich der *OpenGL/ES*-Bibliothek zur Erzeugung von 3D-Grafik. Die meisten dieser Bibliotheken sind bewährten Open-Source-Projekten entnommen.

Auf gleicher Höhe mit den Standardbibliotheken befindet sich die Android-Laufzeitumgebung (*Android Runtime*), welche aus der *Dalvik Virtual Machine* (Dalvik VM) und den *Android Libraries* besteht. Die Dalvik VM charakterisiert dabei im Grunde eine gängige Java VM. Sie wurde jedoch für ressourcenbeschränkte Hardware optimiert und sorgt dafür, dass jede Applikation in ihrer eigenen VM effizient ausgeführt werden kann. Sie bedient sich dabei speziellem Registercode (genannt *dex-Bytecode*, welcher während der Kompilierung mit dem Tool *dx* aus Java-Bytecode gewonnen wird (siehe Abbildung 2.4). Sie wurde von ihrem Entwickler Dan Bornstein nach einem isländischen Dorf benannt, wo einige seiner Vorfahren lebten. Meiner Meinung nach drückt sich in der Namenswahl wohl aber auch eine Analogie zu Java aus, dessen Namen von der indonesischen Insel herrührt. Zusätzlich bieten die *Android Libraries* viele bekannte Pakete der *Java Standard Edition* sowie androideigene Funktionalitäten an.

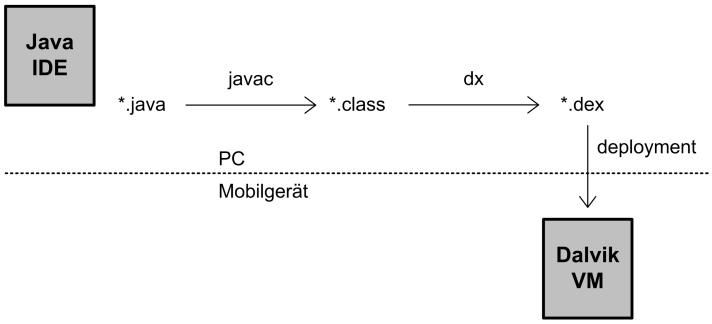

Abbildung 2.4: Erstellungsvorgang des dex-Bytecodes. [1] S. 17

Eine Schicht höher befindet sich das *Application Framework*, welches die Rahmenfunktionen bereitstellt, Anwendungen zu erstellen. Es beinhaltet beispielsweise den *ActivityManager*, der für die Verwaltung der Anwendungs-Lifecycles verantwortlich ist oder den *WindowManager*, der die visuelle Darstellung übernimmt.

Weiters befinden sich hier auch die Komponenten *Package Manager*, *Telephony Manager*, *Resource Manager*, *Content Providers*, *Location Manager* sowie der *Notification Manager*.[1] Auf manche wird in der späteren Sektion *Schlüsselkonzepte* (Abschnitt 2.1.5) näher eingegangen.

Die oberste Schicht repräsentiert die verschiedenen Programme. Dies können die in Android vorinstallierten Applikationen (z.B. Homescreen, Kontakte, Dialer, Webbrowser, u.s.f.) und Programme von Drittanbietern sein. Hier findet die eigentliche „Mensch-Maschine-Interaktion“ statt.[2]

Abbildung 2.5 zeigt exemplarisch die Anwendung *Home*, welche den Desktop eines Android-Systems repräsentiert.

Abbildung 2.5: Standardmäßiger Desktop des Android-Systems.

[1] Vgl. [4] S. 5ff
[2] Vgl. [1] S. 16

2.1.4 Lifecycles

Process Lifecycle

Eine Android-Applikation hat während ihrer gesamten Lebenszeit keinen Einfluss auf ihren Zustand. Dieser wird vom System vorgegeben und bei einer Zustandsänderung mittels entsprechender Methodenaufrufe (sogenannte *Callback*-Methoden) der Applikation mitgeteilt. Dadurch kann die Applikation adäquat auf eine Zustandsänderung reagieren.

Der Grund für dieses Vorgehen ist der Umstand, dass das System versucht, immer reaktiv zu bleiben (also z.B. auf Userevents zu reagieren oder ankommende Anrufe zu verwalten).[1]

Antwortet eine Applikation nicht innerhalb von fünf Sekunden, oder braucht ein *Broadcast Receiver* mehr als zehn Sekunden für die Abarbeitung seiner Aufgabe, wird die *Application Not Responsive* (ANR)- Meldung angezeigt, was es als Entwickler durch vernünftiges Softwaredesign zu verhindern gilt.[2] Abbildung 2.6 zeigt eine solche Meldung.

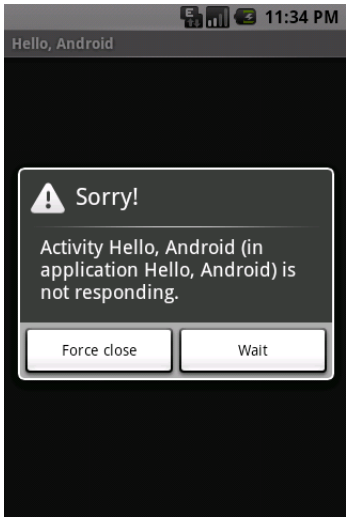

Abbildung 2.6: Meldung *Application Not Responsive*. [21]

Da normalerweise jede Applikation als eigener Prozess in einer eigenen Instanz der VM läuft, kann es passieren, dass dem System die Ressourcen zu knapp werden

[1] Vgl. [4] S. 77f
[2] Vgl. [21]

und es einen Prozess beenden muss.[1] Ein Prioritätssystem regelt dabei, welcher Prozess in einem solchen Fall terminiert werden muss. Diese Prozessprioritäten und -zustände werden in Abbildung 2.7 gezeigt.

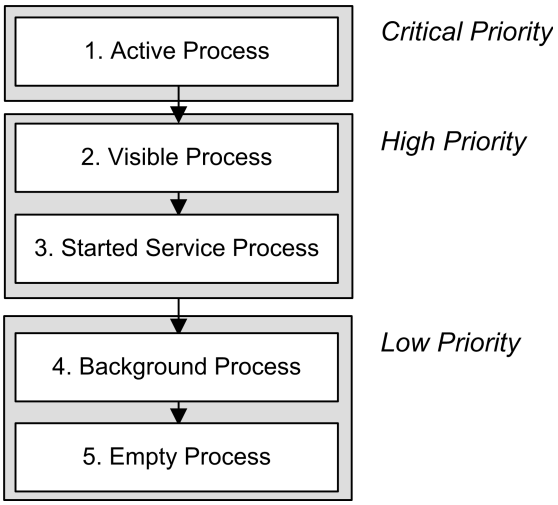

Abbildung 2.7: Prozesszustände und Terminierungsabfolge. [3] S. 51

Demzufolge können Prozesse die Zustände *aktiv*, *sichtbar*, *gestartetes aktives Service*, *Hintergrund* und *nicht aktiv* haben.

Aktive Prozesse sind diejenigen, die gerade mit dem Benutzer interagieren. Dies sind sich im Vordergrund befindende aktive Activities bzw. Activities, Services oder Broadcast Receiver, welche gerade einen Event-Handler (z.B. `onStart()`) ausführen. Derartige Prozesse genießen die höchste, *kritische* Priorität und werden zuletzt terminiert, um Ressourcen freizugeben.

Sichtbare Prozesse hingegen sind Prozesse, die aufgrund teilweiser Verdeckungen (oder Transparenzen) zwar sichtbar aber nicht aktiv sind. Sie werden mit hoher Priorität versehen.

Die nächste Prioritätsstufe bilden *gestartete und aktive Services*. Services werden für fortlaufende Berechnungen verwendet. Obwohl Services normalerweise keine sichbaren Oberflächen haben, werden sie als *vordergründig* eingestuft und damit nur in Ausnahmefällen beendet, wenn Ressourcen *unbedingt* für sichtbare oder aktive Prozesse frei gemacht werden müssen.

Als *Hintergrundprozesse* werden die Prozesse eingestuft, welche im Moment weder

[1] Vgl. [4] S. 77f

aktive Activities, noch gestartete Services haben. Sie werden nach dem „last-seen-first-killed“-Prinzip terminiert.

Die niedrigste Priorität wird *nicht aktiven* Prozessen zugeordnet. Dies sind bereits beendete Prozesse, die zur Performanzsteigerung im Speicher behalten werden, sofern noch Ressourcen vorhanden sind. Sie werden jedoch als erstes verworfen, sobald Ressourcen benötigt werden.[1]

Activity Lifecycle

Eine Activity ist genau ein UI-Screen (siehe exemplarisch der Home-Screen aus Abbildung 2.5) und jede Activity hat wiederum ihren *eigenen* Lifecycle.

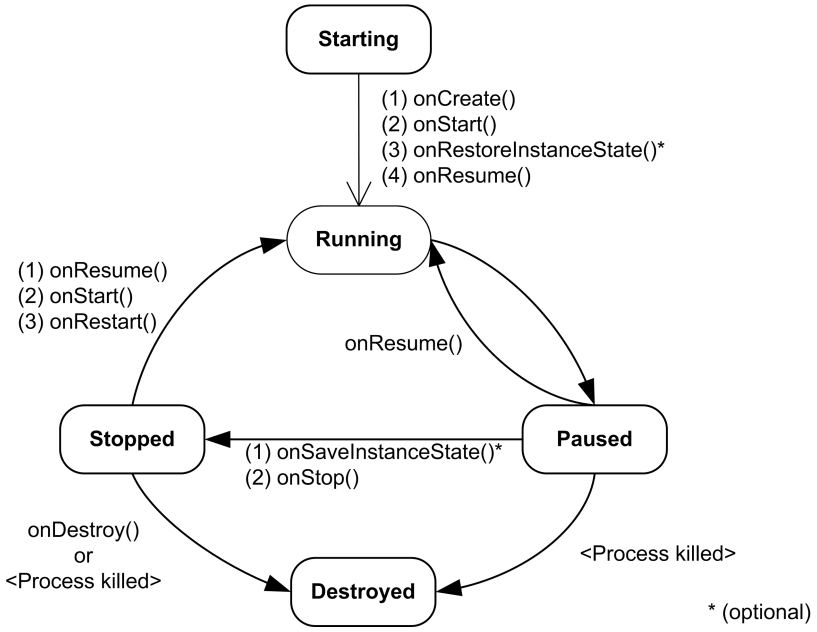

Abbildung 2.8: Lifecycle einer Activity. [2] S. 18

In Abbildung 2.8 ist das Zustandsdiagramm des Lifecycles einer Activity veranschaulicht. Indem man die `onXX()`-*Callback*-Methoden überschreibt, kann man der Activity Anweisungen geben, die jedes Mal dann ausgeführt werden, wenn das System ihren Zustand ändert.

[1] Vgl. [3] S. 51f

Sobald eine Activity gestartet wird, wird ihre `onCreate(Bundle)`-Methode aufgerufen. Hierin initialisiert man normalerweise das GUI und stellt den vorherigen Oberflächenzustand (z.B. ausgewählte Checkboxen) auf Basis des `Bundle`-Parameters wieder her.

Wird die Activity am Display angezeigt, wird `onStart()` gefolgt von `onResume()` ausgeführt. `onResume()` wird auch immer dann ausgeführt, wenn die Activity zur Benutzerinteraktion bereit ist.

Wechselt eine Activity in den Hintergrund, beispielsweise weil eine andere gestartet wurde und vordergründig angezeigt wird, wird die Methode `onPause()` aufgerufen. Hierin sollte man für die „persistente Datenspeicherung“ sorgen.

Ist die Activity nicht sichtbar *und* ist sie nicht mehr in Verwendung, so wird die Methode `onStop()` exekutiert. Sollte das System jedoch einfach den zugehörigen Prozess der Activity beenden, kann es sein, dass diese Methode nicht aufgerufen wird.

Wenn `onRestart()` ausgeführt wird, zeigt dies an, dass eine Activity aus dem gestoppten Zustand wieder angezeigt wird und mit der Methode `onDestroy()` wird das Beenden der kompletten Activity eingeläutet. Auch hier kann es passieren, dass diese Methode nicht aufgerufen wird, nämlich dann, wenn das System aufgrund Ressourcenmangels den Prozess beendet.

Zusätzlich existieren noch die Methoden `onSaveInstanceState(Bundle)` und `onRestoreInstanceState(Bundle)`, welche optional zur *expliziten* Speicherung und Wiederherstellung der Oberfläche verwendet werden können.[1]

Zusammenhang Activity Lifecycle und Process Lifecycle

Eine Applikation besteht aus mindestens einer Activity und *zusätzlich* einem Prozess mit den weiter oben beschriebenen Lifecycles. Jedoch ist ein Activity-Lifecycle nicht an den Process-Lifecycle gebunden. Mit anderen Worten, es kann eine Activity noch im *Application Stack* verweilen, obwohl ihr zugehöriger (aber nicht von ihr abhängiger!) Prozess bereits beendet wurde.

Wird eine Applikation gestartet, werden nach und nach ihre jeweiligen Activities durch den *Activity Manager* auf den *Application Stack* gelegt. Damit wird es möglich, dass das System zur vorherigen Anzeige zurückkehrt, wenn der Benutzer am Gerät die *Zurück-Taste* betätigt.[2]

[1] Vgl. [2] S. 18f
[2] Vgl. [2] S. 16-19

> *„From the user's point of view, it works a lot like the history in a web browser. Pressing back returns them to the previous page.“*[1]

Auf die in dieser Sektion vorweggenommenen Konzepte der Activities, Services und Broadcast Receiver wird im nächsten Abschnitt näher eingegangen.

2.1.5 Schlüsselkonzepte

Wurde in den vorderen Sektionen Androids Architektur vorgestellt, so folgen nun die Schlüsselkonzepte, mit denen jeder Android-Developer vertraut sein sollte. Das sind vorgefertigte Komponenten (so genannte *Building Blocks*), die im System bestimmte Arten von Aufgaben übernehmen und dadurch dem Programmierer die Arbeit enorm erleichtern (können). Namentlich sind das die Konzepte *Activities*, *Services*, *Intents* und *Broadcast Receiver*, *Content Providers*, *Externalized Resources* und *Manifest, Security- und Permissionsystem.*

Activities

Wie bereits im in Abschnitt 2.1.4 angemerkt, handelt es sich bei einer *Activity* um einen einzelnen Bildschirm des Programms, ähnlich einer *Form* in anderen Konzepten. Innerhalb einer Activity können sodann visuelle Komponenten (UI-Elemente wie Textfelder, Buttons, etc.) eingesetzt werden. Würde das nicht gemacht werden, enstünde einfach ein leerer Bildschirm.

Implementiert wird eine Activity, indem man eine Klasse von `Activity` erben lässt:

```
public class MyActivity extends Activity{...}
```

Damit man eine Activity nutzen kann, muss sie im *Manifest* (Abschnitt 2.1.5) registriert werden. Ihr Lifecycle (siehe Abschnitt 2.1.4) beginnt mit dem Erzeugen einer Instanz der Activity.

In der Regel stellen Activities das dar, was für einen Anwender aus seiner Sicht ein Programm ausmacht. Es gibt eine Hauptactivity, die als Einstiegspunkt dient. Von ihr aus ensteht dann der weitere Programmfluss.[2]

Die Begriffe *Activity*, *Screen*, *Bildschirm* und *Form* haben in diesem Kontext alle die gleiche Bedeutung.

[1] [2] S. 16
[2] Vgl. [4] S. 85

Services

Services laufen im Gegensatz zu Activities im Hintergrund, von wo aus sie Daten aktualisieren, auf Ereignisse reagieren oder Daten berechnen. Sie sind die Komponenten, die in einer Applikation die Hintergrundarbeit erledigen. Sie laufen nämlich auch, wenn die eigentliche Applikation inaktiv, nicht sichtbar oder sogar geschlossen ist (vorausgesetzt es sind genügend Ressourcen vorhanden).

Mögliche Kandidaten für Services wären beispielsweise ein Musik-Player, der einmal gestartet, keine Interaktion benötigt, um ein Lied fertig abzuspielen oder ein Erdbebenmonitor, der kontinuierlich die Erdbebendaten herunterlädt, die Informationen verarbeitet und daraufhin gezielte Aktionen setzt. Das könnte z.B. das Starten einer Erdbeben-Activity sein, oder das Abspielen eines Klangs.[1]

Es ist auch möglich, eine Activity mit einem Service zu verbinden. Sie kann dann mit dem Service kommunizieren und zum Beispiel Einstellungen im Service verändern oder Methoden im Service aufrufen.[2]

Indem man eine Klasse von Service erben lässt, wird ein `Service` erstellt:

```
public class MyService extends Service{...}
```

Wie eine Activity muss auch ein Service im *Manifest* (Abschnitt 2.1.5) Erwähnung finden.

Intents, Broadcast Intents und Broadcast Receiver

Wie schon weiter oben (siehe Abschnitt 2.1.3) erwähnt, laufen alle Applikationen in ihrer eigenen Sandbox. Das heißt, jede Applikation ist von allen anderen isoliert und kann von dort nur unter geregelten Bedingungen auf Systemressourcen und Hardware zugreifen. Um auch mit anderen Anwendungen kommunizieren zu können, wurden die Konzepte der *Intents*, *Broadcast Intents* sowie der zugehörigen *Broadcast Receiver* geschaffen.[3]

> *„Intents are used as a message-passing mechanism that lets you declare your intention* [dt: „Absicht“] *that an action be performed, usually with (or on) a particular piece of data.“*[4]

[1] Vgl. [3] S. 250
[2] Vgl. [3] S. 258
[3] Vgl. [4] S. 89
[4] [3] S. 114

Intents stellen in Android die grundlegende Art dar, die Interaktion und Kommunikation zwischen verschiedenen Applikationen bzw. Komponenten durchzuführen. Die Verwendung von Intents, um Aktionen - sogar innerhalb der eigenen Applikation - zu propagieren, „ist ein fundamentales Android-Designprinzip“:[1]

> *„It encourages the decoupling of components, to allow the seamless replacement of application elements. It also provides the basis of a simple model for extending functionality.“*[2]

Intents zum Starten von Activities

Die gewöhnlichste Nutzung von Intents ist das Starten von Activities, entweder *explizit* (die zu startende Klasse wird explizit angegeben) oder *implizit* (z.B. *Öffne Foto X*). In diesem konkreten impliziten Fall würde das System alle Activities suchen, die registriert sind, das angegebene Foto (in Bezug auf Format, etc.) verarbeiten zu können, die passendste Activity auswählen und starten. Der gestarteten Activity steht nun das Foto zur Verfügung und sie würde es, vorausgesetzt sie repräsentiert einen Pictureviewer, am Display anzeigen. Dieser Vorgang ist im Grunde vergleichbar mit gängigen Betriebssystemen, die beim Doppelklick auf ein Foto auch selbst entscheiden, mit welchem Programm sie das Foto öffnen.

Ein anderes Beispiel: Man möchte jemanden anrufen; bevor man diese Funktionalität selbst komplett neu implementiert, kann man auch einen impliziten Intent verwenden:

```
Intent intent = new Intent(Intent.ACTION_DIAL,Uri.parse("tel:123456"));
startActivity(intent);
```

Android löst diesen Intent auf und startet die passendste Activity. In diesem Fall den Dialer.[3]

Intents als PlugIn-System

Wie man erahnen kann, zielt diese Vorgehensweise hauptsächlich darauf ab, Komponenten von deren Applikationen zu entkoppeln. Das vergrößert den Code-Reuse enorm und man braucht das Rad nicht andauernd neu zu erfinden.
So stellt dieses Konzept gleichzeitig ein PlugIn-Modell dar: Das System kann Menüs auf Basis aller für ein Datenset möglicher Activities erstellen. Zur Verdeutlichung wieder das Beispiel mit dem Foto von oben: Anstatt ein Foto zu öffnen, könnte man zur Laufzeit ein Menü generieren, das alle Programme auflistet, die zu diesem Zeitpunkt als befähigt registriert sind, mit einem Foto umzugehen. Das mag heute nur

[1] Vgl. [3] S. 114
[2] [3] S. 114
[3] Vgl. [3] S. 114f

sein, es zu betrachten oder es als eMail zu versenden. Doch bereits Morgen könnte es sein, dass man die Fotos auch direkt an einen Fotodrucker schicken möchte. Dann bräuchte man nur eine Activity mit dieser Funktionalität registrieren und ab diesem Zeitpunkt würde das Menü auch die neue Funktionalität anbieten.[1]

Broadcast Intents und Broadcast Receiver

`Intents` werden auch als systemweiter Nachrichtendienst verwendet. Diese sogenannten `Broadcast Intents` können von jeder Applikation mit einem `Broadcast Receiver` abgefangen und verarbeitet werden. Damit lassen sich event-gesteuerte Programme auf Basis jeglicher verfügbarer Intents (Systemintents oder Intents von Applikationen von Drittanbietern) kreieren.

Das Erstellen und Senden von Broadcast Intents erfolgt folgendermaßen:

```
Intent intent = new Intent("com.paad.action.EVENT_XY_OCCURED");
sendBroadcast(intent);
```

`Broadcast Intents` werden innerhalb Androids für alle möglichen Systemevents benutzt: falls sich der Batterie-Status ändert, ein Anruf eingeht oder eine SMS ankommt u.s.w. Indem man zum Beispiel den *incoming call* - Broadcast abfängt, könnte man die Lautstärke abhängig vom Anrufer einstellen.

`Intents` helfen, Applikationen offener zu gestalten. Indem man Events durch einen Intent ankündigt, gibt man auch Drittentwicklern die Chance, auf diese zu reagieren, ohne dass sie die originale Applikation verändern müssen.
Innerhalb einer Applikation kann man Broadcast Intents abfangen und damit native Applikationen ersetzen oder erweitern bzw. auf System- und Programmevents in eigener Weise reagieren.[2]

Intent mit Daten versehen

Um einem Intent auch Daten anzufügen, ist es möglich, einen *Uniform Resource Identifier* (URI) zu spezifizieren oder mit der Methode `putExtra` zusätzliche „Key-Value-Pairs“ zu setzen. Mit der Methode `setType` kann man zusätzlich einen MIME-Type setzen, was aber nur gemacht werden sollte, wenn kein URI angegeben wird, da ein URI bereits *implizit* einen MIME-Type charakterisiert.

Beispielsweise würde

```
Intent intent = new Intent(Intent.ACTION_VIEW, "file:///tmp/picture.jpg");
```

den Intent zum Anzeigen eines Fotos definieren, wohingegen

[1] Vgl. [3] S. 130
[2] Vgl. [3] S. 132f

```
Intent intent = new Intent("com.paad.action.EVENT_XY_OCCURED");
intent.putExtra("where", "TU-Vienna");
```

einen Intent mit der ID `com.paad.action.EVENT_XY_OCCURED` und der zusätzlichen Information `where = TU-Vienna` definiert.[1]

Exkurs: Projekt OpenIntents

Damit man öfters genutzte, abgegrenzte Funktionalitäten nicht immer wieder neu implementieren muss, hat sich das OpenIntents-Projekt gebildet. Es bietet Komponenten an, die nativ von Android (noch?) nicht angeboten werden, wie beispielsweise einen FilePicker.[2]

Content Providers

Der Zugriff auf Daten ist innerhalb einer Applikation wegen des *Sandboxing* stark beschränkt. Keine Applikation kann und darf Daten einer anderen Applikation lesen oder ändern. Für gewisse Daten ist jedoch genau dies erwünscht und deshalb wurden die *Content Providers* entwickelt.[3]Sie kapseln die dahinterliegenden Datenquellen und stellen sie über ein generisches Interface systemweit zur Verfügung. Damit wird es möglich, den *application layer* vom *data layer* zu entkoppeln.

Content Providers bieten volle Zugriffskontrolle, Query/Read/Write-Access und werden mit einem einfachen URI-Modell systemweit zugänglich gemacht. Der Aufbau einer URI eines Content Providers gleicht dem einer Web-URL und sieht wie folgt aus:

```
content://com.<CompanyName>.provider.<AppName>/<DataPath>
```

Um auf einen Content Provider zuzugreifen, verwendet man das `ContentResolver`-Objekt, das die folgenden Methoden bereitstellt:

- `query()` zum Abfragen von Content. Eine Abfrage funktioniert ähnlich einer SQL-Abfrage und gibt ein `Cursor`-Objekt zurück.
- `insert()` fügt einen Eintrag hinzu und gibt die URI zum eingefügten Eintrag zurück.
- `delete()` löscht einen oder mehrere Einträge.
- `update()` verändert einen oder mehrere bestehende Einträge und gibt die Anzahl der veränderten Einträge zurück.

[1] Vgl. [3] S. 132f
[2] Vgl. [22]
[3] Vgl. [4] S. 74

Die wichtigsten nativen Android Content Providers sind *Contacts* zum Zugriff auf die Kontakte, *MediaStore* zum Zugriff auf Multimediaressourcen und *Settings* zum Einsehen und Ändern von Geräteeinstellungen.

Eigene `Content Providers` werden mittels

```
public class MyProvider extends ContentProvider { }
```

implementiert und müssen im *Manifest* (siehe Abschnitt 2.1.5) registriert werden.

Meist kapseln sie eine applikationsinterne SQLite-Datenbank und stellen sie dadurch über die Applikationsgrenzen hinaus bereit.[1]

Externalized Resources

Unter Android ist es möglich, alle Nicht-Code-Ressourcen, in einem eigenen Segment des Projekts zu verwalten. Dies geschieht im Ordner `res` (siehe Abbildung 2.9) und bezieht sich neben Ressourcen wie Strings, Colors, Arrays, Drawables (Icons, Images) und Animations auch auf das Definieren der Layouts von Benutzeroberflächen.

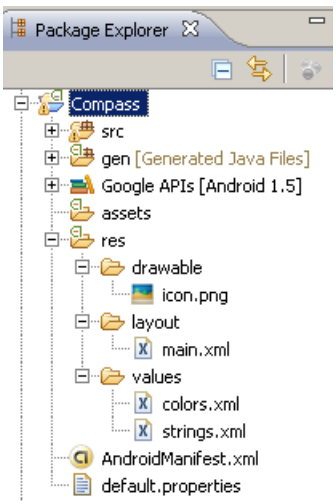

Abbildung 2.9: Projektstruktur mit Ordner "res", wo die externen Ressourcen verwaltet werden.

Der Ordner `res/anim` speichert Animationen, `res/drawable` Bilddaten, `res/layout` GUI-Layouts und der Ordner `res/values` verschiedene Werte für Ar-

[1] Vgl. [3] S. 189-196

rays, Farben, Dimensionen, Zeichenketten und Stile. Beliebige XML-Dateien werden in `res/xml` verwaltet.[1]

Dynamische Anpassung und Internationalisierung

Indem man den verschiedenen Ressourcen-Ordnern bestimmte Zeichenketten anhängt, kann man alternative Ressourcen definieren, die abhängig von äußeren Zuständen wie Sprache, Region und Hardware dynamisch eingebunden werden. Beispielsweise ist es damit leicht möglich, internationalisierte Versionen seines Programms zu erstellen oder verschiedene Layouts für Hoch- oder Querformat zu definieren. Die gerade passendste Ressource wird sodann zur Laufzeit in das Programm eingebunden, beispielsweise wenn sich die Bildschirmausrichtung des Geräts von Hochformat auf Querformat ändert.

Weitere mögliche Kennzeichner zur dynamischen Anpassung gibt es für Pixeldichte (dpi), Bildschirmauflösung, Touchscreentyp (Fingerbedienung oder Stiftbedienung), Tastaturverfügbarkeit, Tastaturtyp und aktuelle Art der Oberflächennavigation (Dpad, Trackball, Wheel, u.a.). Teilweise können diese Kennzeichner auch miteinander kombiniert werden, z.B. legt der Ordnername `drawable-land-en-rGB-320x240` die drawable-Ressourcen für querformat, englische Sprache, Region England und eine Auflösung von 320x240 Pixel fest.[2]

Verwendung der externen Ressourcen

Falls die *Eclipse IDE* zur Entwicklung verwendet wird, wird automatisch eine Datei namens `R.java` im Ordner `/gen` angelegt, die alle externen Ressourcen über eine Klasse verwaltet. Dabei bekommt jede Ressource eine eindeutige ID zugewiesen und kann damit im Code wie folgt referenziert werden:

```
setContentView(R.layout.main);
Drawable icon = getResources().getDrawable(R.drawable.appicon);
CharSequence seq = getString(android.R.string.httpErrorBadUrl);
```

Ressourcen beginnend mit `android.R.` charakterisieren Systemressourcen. Mit dem `@`-Operator wird innerhalb externer Ressourcen auf andere externe Ressourcen referenziert (z.B. `<EditText android:text="@string/message"/>`) und um ein Element explizit mit einer referenzierbaren ID zu versehen, verwendet man `@+id/myID`, zum Beispiel `<EditText android:id="@+id/myTextbox"/>`.[3]

[1] Vgl. [4] S. 79-82
[2] Vgl. [3] S. 63f
[3] Vgl. [4] S. 82

Manifest und Security- und Permissionsystem

Android ist quelloffen, Programme haben durch die APIs Zugriff auf die Hardware und können auf unabhängigen, verschiedenen Wegen verbreitet werden. Aus diesem Grund spielt auch die Sicherheit eine wesentliche Rolle.

Nicht nur, dass eine Applikation wegen der Sandbox sowieso nur Zugriff auf ihre eigenen Daten hat (siehe Abschnitt 2.1.3), das System verbietet standardmäßig auch den Zugriff auf verschiedene Services und Funktionalitäten. Damit werden Programme gezwungen, zuerst eine Nutzungserlaubnis einzuholen. Dies passiert während der Installation eines Programms, wo der User die vom Programm angeforderten Rechte einsehen und gewähren muss.[1]

Das gesamte Programm-Management geschieht in der XML-Datei `AndroidManifest.xml`. Das `<manifest>`-Tag stellt den Wurzelknoten dar und nimmt alle anderen Tags auf. Es beinhaltet u.a. Metainformationen (Package-Name, Versionsnummer, u.a.) der gesamten Anwendung. Das `application`-Element definiert den Programmtitel und sein Icon und kapselt alle in der Applikation verwendeten Komponenten. Diese Komponenten sind `<activity>`, `<service>`, `<provider>` und `<receiver>`. Innerhalb dieser Tags wird u.a. auf die konkreten Klassennamen der Komponenten referenziert oder sonstige Einstellungen festgelegt. So kann beispielsweise ein `Intent-Filter` angelegt werden, der das Starten einer Komponente nur von einem bestimmten Intent zulässt, oder ein `android:permission`-Tag, das die Zugriffsrechte für Content Provider festsetzt. Auszüge einer typischen Manifest-Datei zeigt Listing 2.1.

```
1  <manifest package="com.paad.earthquake" android:versionCode="1">
2      <application android:icon="@drawable/icon" android:label="
           Earthquake-Viewer">
3          <activity android:name="myActivity">
4              <intent-filter>
5                  <category android:name="android.intent.category.
                       LAUNCHER"/>
6              </intent-filter>
7          </activity>
8          <provider android:name="myProvider"/>
9          <service android:name="myService"/>
10     </application>
11     <uses-permission android:name="android.permission.INTERNET"/>
12 </manifest>
```

Listing 2.1: Auszüge einer typischen Manifest-Datei. [3] S. 267ff

Damit das System einer Anwendung bestimmte kritische Ressourcen gewähren

[1] Vgl. [3] S. 36

kann, müssen weiters Erlaubnisse erteilt werden. Diese werden mittels `Uses-Permissions` angefordert und vom System erteilt. So existieren beispielsweise die Rechte auf GPS, Kontaktdaten, das Internet, u.s.f. zuzugreifen. Auch diese Anforderungen werden dem Benutzer während der Installation der Applikation zur Bestätigung vorgelegt. Damit wird der Benutzer in transparenter Weise informiert, welche Ressourcen das Programm verwendet.[1]

2.1.6 Hilfreiche APIs des SDK

Wurden in der obigen Sektion bereits einige Schlüsselkonzepte vorgestellt und befinden sich in den „Core Libraries“ (siehe Runtime Environment Abschnitt 2.1.3) einige aus Java Standard Edition (Java SE) bekannte Programmierschnittstellen, so sollen in diesem Abschnitt noch weitere hilfreiche, meist androideigene APIs des Software Development Kit beschrieben werden:

Telephony

Package Name: `android.telephony`

Funktionalität: Bietet mit dem `TelephonyManager` Zugriff auf Telefon-Informationen (Netzwerktyp, Netzwerkzelle, Status, u.a.) und mit dem `SmsManager` Zugriff auf SMS.[2]

Anmerkung: Obwohl Android die grundsätzliche Möglichkeit bietet, native Programmteile durch eigene zu ersetzen, erlaubt es das derzeitige SDK wegen Sicherheitsbedenken nicht einen eigenen *Dialer* zu erstellen.[3]

Media

Package Name: `android.media`

Funktionalität: Stellt Objekte zur Verfügung, die (Multi)Mediadaten verarbeiten. Diese sind u.a. `MediaPlayer`, `MediaRecorder`, `FaceDetector`, `ToneGenerator`, `AudioManager` und `RingtoneManager`.[4]

[1] Vgl. [4] S. 74-76
[2] Vgl. [23]
[3] Vgl. [3] S. 334
[4] Vgl. [23]

Location

Package Name: `android.location`

Funktionalität: Die Location API bietet mit dem `LocationManager`, den `LocationProviders` und den `Geocoders` essenzielle Funktionen für *Location Based Services* (LBS) an. Der `Location Manager` ist dafür zuständig, die geographische Position des Geräts herauszufinden und diese bei Bedarf auch regelmäßig neu zu bestimmen. Ein `Location Provider` bestimmt die dafür notwendige Technologie (GPS- oder netzwerkbasiert).

```
1  // set creteria for location provider
2  Criteria crit = new Criteria();
3  crit.setAccuracy(Criteria.ACCURACY_FINE);
4  crit.setPowerRequirement(Criteria.POWER_LOW);
5
6  // get location-service
7  LocationManager lm =
8      (LocationManager)getSystemService(Context.LOCATION_SERVICE);
9  // get best provider based on criteria
10 String provider = lm.getBestProvider(crit, true);
11 // request locationupdates,
12 updateFreq = 30000; // in miliseconds, e.g. every 30 seconds
13 updateDistance = 100; // e.g. every 100 meters
14 lm.requestLocationUpdates(provider, updateFreq, updateDistance,
       locationListener);
15
16 // set location listener
17 private final LocationListener locationListener = new
       LocationListener() {
18     @Override
19     public void onProviderDisabled(String provider) { }
20         @Override
21         public void onProviderEnabled(String provider) { }
22         @Override
23         public void onStatusChanged(String provider, int status,
               Bundle extras) {}
24         @Override
25         public void onLocationChanged(Location location) {  }
26     };
```

Listing 2.2: Verwendung der Location API.

Die grundsätzliche Verwendung der Location API wird in Listing 2.2 dargestellt. Zuerst werden Kriterien bezüglich gewünschter Genauigkeit, Energieverbrauch, etc. festgelegt und vom `LocationManager` wird der beste Anbieter ermittelt. Danach werden mittels Methode `requestLocationUpdates` fortwährende Standortaktualisierungen angefordert. Dabei geben die Parameter `updateFreq` und `updateDistance` ein Zeit- und Distanzintervall an zwischen denen dem `locationListener` ein neuer Standort mitgeteilt wird. Indem man

im `LocationListener` die `onXX()`-Methoden überschreibt, kann auf verschiedene Änderungen reagiert werden.

Weiters bietet ein `Geocoder` die Möglichkeit, anhand von Breiten- und Längengraden die zugehörige Adresse herauszufinden (*Reverse Geocoding*) und umgekehrt (*Forward Geocoding*).[1]

Diese Daten können in Verbindung mit einer *MapView* (siehe Abschnitt 2.6) visuell dargeboten werden. Für genauere technische Hintergründe bezüglich Tracking und LBS sei auf Abschnitt 2.2 verwiesen.

Sensor Hardware

Package Name: `android.hardware`

Funktionalität: Dieses Paket beinhaltet Programmierschnittstellen zum Zugriff auf die Kamera (`Camera`) und weitere Sensoren (`Sensors`), die durch den `SensorManager` verwaltet werden. Damit können die Messdaten von einem *Beschleunigungssensor* (Accelerometer), *Helligkeitssensor*, *Thermometer*, *Kompass*, *Orientierungssensor*, *Drucksensor*, *Gyroskop* bzw. *Näherungssensor* herausgelesen werden.[2]

Besonders interessant für das dieser Arbeit zugrunde liegende Projekt ist dabei der Beschleunigungssensor, welcher die Beschleunigungen in drei Richtungen (auf/ab, links/rechts, vor/zurück) misst und die Basis für den Schrittzähler legt (siehe Abschnitt 2.3. Weiters verwende ich auch den Kompass, ein Sensor für das magnetische Feld in Richtung der drei Achsen und den Orientierungssensor zum Detektieren, ob das Gerät hoch- oder querkant gehalten wird.

Um auf Änderungen in den Sensordaten dynamisch reagieren zu können, muss ein `SensorEventListener` registriert und darin die Methode `onSensorChanged` überschrieben werden. Die Frequenz mit der die Daten vom Sensor erhoben werden, wird beim Registrieren des `SensorEventListener` mit angegeben. Das Grundgerüst, um auf Sensordaten des Akzelerometers zuzugreifen wird in Listing 2.3 gezeigt.

Bei der Entwicklung von Programmen ist zu beachten, dass nicht jedes Endgerät all diese Sensoren eingebaut haben muss.[3]

[1] Vgl. [3] S. 208-222
[2] Vgl. [23]
[3] Vgl. [3] S. 319-ff

```
1  // get sensormanager
2  SensorManager sm =
3      (SensorManager)getSystemService(Context.SENSOR_SERVICE);
4  // register sensoreventlistener
5  SensorEventListener accelerometerListener = new SensorEventListener
       () {
6      @Override
7      public void onSensorChanged(SensorEvent event) { }
8      @Override
9      public void onAccuracyChanged(Sensor sensor, int accuracy) { }
10 };
11 // request accelerometer-sensor
12 Sensor sensor = sm.getSensorList(Sensor.TYPE_ACCELEROMETER).get(0);
13 if (sensor != null){
14     // register listener to sensor with normal update-frequency
15     sm.registerListener(accelerometerListener, sensor,
16         SensorManager.SENSOR_DELAY_NORMAL);
17 }
```

Listing 2.3: Verwendung des Beschleunigungssensors.

Network

Package Name: `android.net`

Funktionalität: Das zentrale Objekt dieser API ist der `ConnectivityManager`. Seine Verantwortlichkeiten liegen in der Überwachung der Netzwerkverbindungen (WiFi, GPRS, UMTS, ...) und im Benachrichtigen des Systems, sobald Änderungen bezüglich der Verbindungen auftreten.[1]

Im Package `android.net.wifi` gibt es weiters noch den `WifiManager`. Mit ihm kann auf sämtliche Aspekte von WLan zugegriffen werden.

Bezüglich Bluetooth wird zwar die Verwendung von Stereoheadsets (Profil A2DP) und Headsets (Profil SCO) unterstützt, die in Beta-Versionen noch verfügbare Unterstützung einer Bluetooth API wurde jedoch in der finalen Release 1.0 wieder fallengelassen. Dazu Nick Pelly, einer der dafür zuständigen Entwickler im „Developers Blog“:

> *„The reason is that we plain ran out of time. The Android Bluetooth API was pretty far along, but needs some clean-up before we can commit to it for the SDK. Keep in mind that putting it in the 1.0 SDK would have locked us into that API for years to come.“*[2]

Anmerkung: Bis hin zur aktuellen Version 1.6 (Stand 20.10.2009) gibt es seitens

[1] Vgl. [23]
[2] [24]

Androids noch keine programmierbare Bluetooth-Unterstützung. Diese soll jedoch mit Version 2.0 eingeführt werden.[1]

SQLite Datenbank

Package Name: `android.database.sqlite`

Funktionalität: Bietet eine API, um in Applikationen Datenbanken zur persistenten Datenhaltung anzulegen. Derartige Datenbanken können mittels `Content Providers` (Abschnitt 2.1.5) abstrahiert über Applikationsgrenzen hinweg zur Verfügung gestellt werden.[2]

2.1.7 Entwicklertools, Signation und Deployment

Entwicklertools

Das Android SDK bietet eine Fülle an Hilfsprogrammen an, die die Android-Applikations-Entwicklung enorm erleichtern. Dazu zählen das automatische *Java und Dalvik Build System*, das aus Java-Byte-Code den für Androidapplikationen obligaten dex-Byte-Code generiert, die *Android Debug Bridge* (ADB) zum Debuggen von der Kommandozeile aus und das *Dalvik Debug Monitor Service* (DDMS), „eine graphisch orientierte Debugging-Umgebung speziell für Android und die Dalvik VM“. Weiters bietet *Traceview* ein Werkzeug, alle Methodenaufrufe und ihre darin verbrauchten Zeiten aufzulisten und mit *Logcat* kann auf ausgedehnte Log-Daten des Systems zugegriffen werden.[3]

All diese Tools können ohne zusätzliche Programme verwendet werden und sind standardmäßig im Android SDK enthalten. Um den Komfort während des Entwicklungsvorgangs zu erhöhen, wird aber auch ein Plugin für die bewährte *Eclipse IDE*, namentlich die *Android Development Tools* (ADT), angeboten. Dieses Plugin bindet die oben erwähnten Tools nahtlos in die IDE ein.

Signation und Deployment

Sobald eine Applikation fertig entwickelt und gut getestet ist, geht es daran, sie zu distribuieren und auf Endgeräten zu installieren (engl. *Deployment*). Aus Sicher-

[1] Vgl. [26]
[2] Vgl. [3] S. 176
[3] Vgl. [5] S. 57f

heitsgründen schreibt das Android-System vor, dass jede Anwendung mit einem Zertifikat signiert sein muss, um überhaupt installiert werden zu können. Der Vorgang der Signation wird in Abbildung 2.10 illustiert:

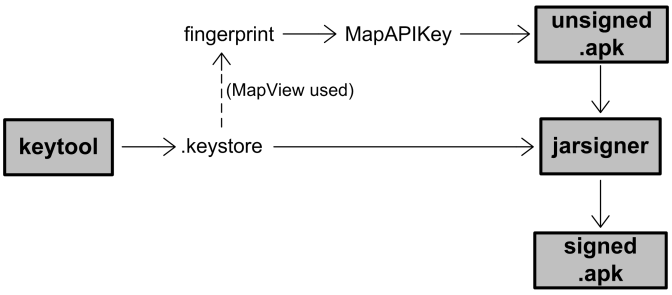

Abbildung 2.10: Signieren einer Applikation.

Man generiert einen Keystore (ein public/private Key-Paar, welcher das Zertifikat repräsentiert) und signiert damit die eigentliche Applikation.

Um das Zertifikat zu generieren, gibt man diesen Befehl in der Kommandozeile ein:

```
keytool -genkey -v -keystore <myAppName>.keystore -alias <myAppKey>
-keyalg RSA -validity 10000
```

Zur Erzeugung wird das Programm `keytool` vom *Sun JDK* verwendet. Der Parameter `<myAppName>.keystore` gibt den Namen des Keystores an, worin die Zertifikatinformationen gespeichert werden sollen, `<myAppKey>` definiert einen alternativen Namen für den Schlüssel und `RSA` die Methode, diese zu generieren. Die abschließende Zahl stellt die Gültigkeit des Zertifikats (in Tagen) ein. Google empfiehlt dabei, diese auf mindestens 25 Jahre zu setzen, weil der *Android Market* eine Gültigkeit mindestens bis zum 22. Oktober 2033 (genau 25 Jahre ab dem ersten Tag des Marketstarts) voraussetzt.

Verwendet man in seiner Applikation auch `MapViews` (Oberflächenelemente zur Anzeige von *Google Maps*-Daten), muss man von Google einen Mapschlüssel einholen, der an den MD5-Fingerprint des Zertifikats gebunden ist. Dazu verwendet man den Befehl

```
keytool -list -alias <myAppKey> -keystore <myAppName>.keystore
```

und ruft mit dem so erhaltenen Fingerprint den MapAPI - Schlüssel im Internet[1] ab. Dieser muss vor dem Signieren in jeder verwendeten `MapView` eingetragen werden.

[1] siehe http://code.google.com/android/maps-api-signup.html

Zur eigentlichen Signation der Anwendung muss anschließend eine unsignierte Version exportiert (in Eclipse: Rechtsklick auf das Projekt - Android Tools - Export unsigned Application Package...) und diese *.apk*-Datei mittels

```
jarsigner -verbose -keystore <myAppName>.keystore myApplication.apk
myAppKey
```

signiert werden.

Der Sinn der Verwendung von Zertifikaten liegt darin, dass „Applikationen, die vorgeben von einem gewissen Entwickler zu sein, das auch wirklich sind." Daher können mehrere Applikationen mit dem selben Zertifikat signiert werden. [1]

Abbildung 2.11: Typische Möglichkeiten der Anwendungsdistribution.

Ein fertiges Programm stellt somit unter Android *eine* `.apk`-Datei dar, in welche sämtliche Komponenten hineingepackt sind. Typischerweise werden solche Dateien zur Distribution einfach im Web als Download angeboten oder im *Android Market* veröffentlicht. Lädt man eine derartige Datei direkt unter Android herunter, wird sie auch automatisch installiert. Ansonsten bietet der Android Market eine zentrale Plattform zur Softwaredistribution (siehe Abbildung 2.11). Der Market ist von einem Android-Gerät aus direkt zugänglich und durchsuchbar und erleichtert einerseits den Usern den Zugang zu neuen Programmen, andererseits auch den Entwicklern den Zugang zu ihren Märkten.[2] Betrieben wird der Market proprietär von der Firma Google. Er ist nicht Teil des Betriebssystems Android, damit nicht opensource und auch nur auf Geräten mit inkludierten Google-Diensten verfügbar.[3] Will man als Entwickler seine Programme im Market anbieten, muss man einmalig Eur 25.- für eine Mitgliedschaft bezahlen. Dann ist es aber auch möglich, seine Programme zu verkaufen, wobei in diesem Fall Google 30% (Stand 20.10.2009) der Einnahmen einbehält.[4]

[1] Vgl. [5] S. 87-97
[2] Vgl. [4] S. 51ff
[3] Vgl. [20]
[4] Vgl. [25]

2.1.8 Zusammenhang Android und Java

Anwendungen für Android werden in Java geschrieben. Als Basis dient Java Version 5.0 und legt damit auch größtenteils die Programmiereigenschaften fest. So wird Javacode im Sprachlevel der Version 5 standardmäßig unterstützt. Auch die Datentypen `byte`, `char`, `short`, `int`, `long`, `float`, `double`, `Object`, `String` und `array` sind verfügbar. Einzig Fließkommazahlen werden aufgrund von Hardwareeinschränkungen meist emuliert und sollten daher spärlich verwendet werden, da sie wegen der Emulation langsamer verarbeitet werden. Weiters werden Multithreading, Synchronisation und Reflection unterstützt, wobei Reflection aus Performanzgründen nicht verwendet werden soll. Des weiteren wird während der Garbage-Collection auch die automatische Objekt-Finalisation durchgeführt. Durch die Ressourcenbeschränktheit der Geräte sollte jedoch in Erwägung gezogen werden, Objekte explizit freizugeben, sobald sie nicht mehr benötigt werden.[1]

Neben vielen Java SE 5.0 - Bibliotheken werden von Android auch `org.apache.http`, `org.json`, `org.xml.sax`, `org.xmlpull.vl` unterstützt. Nicht unterstützt werden allerdings einige APIs, die auf einem mobilen Gerät unter Android „einfach keinen Sinn gemacht hätten“, so zum Beispiel `javax.print` oder `javax.swing`.[2]

2.1.9 Alternative Plattformen

Wird in dieser Arbeit der grundlegende Fokus auf die Plattform Android gelegt, so soll erwähnt werden, dass es auch andere Plattformen für mobile Endgeräte gibt, welche ähnliche Funktionalität bieten. Die bekannteren dieser Plattformen sind *Apple iPhone OS*, *Java ME*, *Microsoft Windows Mobile*, *Palm WebOS*, *RIM Blackberry* und *Symbian OS*.

Apple iPhone/iPod OS

Das *iPhone OS* von der Firma Apple ist das Betriebssystem für ihr Mobiltelefon *iPhone*. Es wird proprietär entwickelt, bietet mit einem öffentlichen *iPhone SDK* jedoch die Möglichkeit, selbst Programme für iPhones zu schreiben. Diese können allen iPhone-Benutzern im *App Store* angeboten werden.[3]

[1] Vgl. [2] S. 207f
[2] Vgl. [2] S. 209f
[3] Vgl. [27]

Java ME

Java ME oder *J2ME* (Java Platform 2 Micro Edition) ist eine Spezifikation, um Java-Applikationen auf embedded Systems (z.B. Mobiltelefone, Settop-Boxen, u.ä.) zu ermöglichen. Grundlage bildet auf jedem Gerät eine speziell dafür entwickelte virtuelle Maschine (VM), welche die in *Konfigurationen* und *Profilen* spezifizierten Funktionen hardwareabstrahiert zur Verfügung stellt.[1]

Microsoft Windows Mobile

Auch Microsoft bietet ein Betriebssystem für mobile Plattformen an. Dieses präsentiert sich optisch sehr ähnlich dem Desktopbetriebssystem Windows, läuft aber auf anderen Rechnerarchitekturen. Eigene Programme können mit der *Microsoft Win32 API* oder unter Verwendung des *.NET Compact Framework 2.0* erstellt werden.[2]

Palm webOS

Palm webOS ist der Nachfolger von *Palm OS* und stellt ein proprietäres Betriebssystem für Smartphones dar. Es ist multitasking-fähig und basiert auf einem Linux-Kernel. Mit dem SDK *Mojo Application Framework* können eigene Programme entwickelt werden.[3]

RIM BlackBerry OS

Die Firma *Research In Motion* (RIM) bietet unter dem Namen *Blackberry OS* ein proprietäres Multitasking-Betriebssystem für seine Blackberry-Produktfamilie. Auch unter diesem Betriebssystem ist es möglich, eigene Programme zu erstellen.[4]

Symbian OS

Auch das *Symbian OS* ist ein Multitasking-Betriebssystem für mobile Geräte. Wie bereits erwähnt (siehe Abschnitt 2.1.1) ist es beabsichtigt, die verschiedenen verwandten Betriebssysteme *Symbian OS*, *S60*, *UIQ* und *MOAP(S)* in

[1] Vgl. [28]
[2] Vgl. [29]
[3] Vgl. [30]
[4] Vgl. [31]

ein einziges zusammenzuführen und daraufhin unter einer open-source Lizenz zu veröffentlichen.[1]Für Symbian OS gibt es mehrere SDKs, welche es unter anderem erlauben, Applikationen zu schreiben (*Application Development SDK*), oder an der Weiterentwicklung des Betriebssystems selbst mitzuarbeiten (*Product Development Kit*).[2]

Übersicht

Plattform	Hersteller	Open-Source	SDK
Android	OHA	ja	ja
iPhone/iPod OS	Apple	nein	ja
Java ME	Sun	nein	ja
Windows Mobile	Microsoft	nein	ja
webOS	Palm	nein	ja
BlackBerry OS	RIM	nein	ja
Symbian OS	Symbian Foundation	angekündigt	ja

Tabelle 2.1: Übersicht über Plattformen mobiler Geräte.

Tabelle 2.1 fasst die wichtigsten Eigenschaften dieser Systeme zusammen. Es zeigt sich, dass sämtliche aktuelle Plattformen interessierten Entwicklern Software Development Kits (SDK) anbieten um selbst Programme zu schreiben. Doch obwohl die meisten SDKs kostenlos erhältlich sind, gehen die Vorstellungen über Kosten bezüglich späterer Software-Distributionen weit auseinander. So kostet beispielsweise bei Apple die Mitgliedschaft als Entwickler jährlich $99 USD und Anwendungen können nur über Apples proprietären AppStore veröffentlicht werden und von etwaigen Einnahmen werden 30% abgezogen.[3] Etwas anders verhält sich dieser Sachverhalt bei Android. Hier können Programme ohne Kosten erstellt und verbreitet werden. Das Anbieten über den Android Market kostet als Entwickler einmalig 25 EUR, erwirtschaftete Einnahmen müssen aber auch zu 30% abgegeben werden.[4] Java ME ist in dieser Hinsicht nicht vergleichbar, da es nur eine Spezifikation darstellt, wie eine Java VM beschaffen sein muss, um J2ME-Code auszuführen. Es ist kein komplettes Betriebssystem, wie die anderen hier vorgestellten Plattformen.

[1] Vgl. [32]
[2] Vgl. [33]
[3] Vgl. [27]
[4] Vgl. [25]

2.2 Tracking und Location Based Services

Tracking oder *Ortung* bezeichnet das Ausfindigmachen der Position, an der man sich gerade aufhält. Diese Position ist meist die Basis für sogenannte *Location Based Services* (LBS), welche abhängig von der automatisch ermittelten Position verschiedene Informationsdienste anbieten.

Innerhalb der Forschungsrichtung *Mobile Spatial Interaction* (MSI) beschäftigt man sich dabei mit vier wichtigen Feldern, die für LBS große Relevanz haben: „Accessing information attached to physical places“, „Navigation and Wayfinding“, „Adding content to physical places or objects“, und „Mobile virtual and augmented reality“. Insbesondere zeigt sich, dass positionssensitive mobile Geräte eine sehr passende Form bilden, mit Geoinformationen zu interagieren. Sie können somit als Brücke zwischen virtuellem und realem Raum angesehen werden.[1]

2.2.1 Tracking-Methoden

Ein LBS setzt immer eine einigermaßen akurate Positionsbestimmung voraus. Obwohl es dafür viele verschiedene technische Ansätze gibt, sind für LBS meist lediglich *Radiolocation Techniques* und *Satellite-Based Techniques* relevant.

Die grundsätzliche Idee der *Radiolocation Techniques* ist es, die Position eines Geräts anhand der Funksignale zu bestimmen, die zwischen dem Gerät und einigen fixen Funkstationen gesendet werden. Diese Methoden können wiederum in *netzbasiert* und *gerätebasiert* unterteilt werden, je nachdem, wo genau die Positionsbestimmung stattfindet und erreichen eine Genauigkeit zwischen 50 und 100 Meter (Stand Jahr 2004). Eine ungenauere, aber in jedem Mobilnetz funktionierende Methode ist die simple Verwendung der Cell IDs. Mithilfe von Wissen über die Netzbeschaffenheit, kann der ungefähre Standort anhand der ID ermittelt werden. Die Genauigkeit hängt dabei von der Mobilfunkzellengröße ab.

Innerhalb der *satellitenbasierten Verfahren* ist die gebräuchlichste Form die Positionsbestimmung mittels *Assisted GPS* (A-GPS). Dies ist ein Verfahren, wo dem GPS-Empfänger fehlende Informationen (z.B. aufgrund zu weniger sichtbarer Satelliten in einer schmalen Gasse) über ein Mobiltelefonnetz zur Verfügung gestellt werden können.

> *„Originally, GPS by itself was not meant for indoor or urban positioning, but the concept of AGPS can be harnessed for that purpose by*

[1] Vgl. [8]

celluar networks [...]“[1]

Diese Methodik wird oftmals auch als *hybrides* Verfahren zur Positionsbestimmung bezeichnet. Abbildung 2.12 zeigt den strukturellen Aufbau von A-GPS.[2]

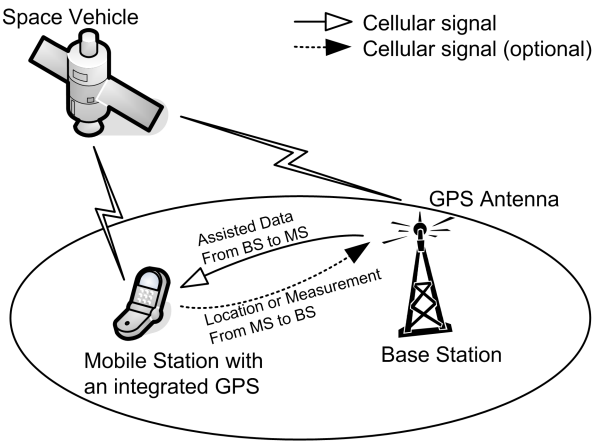

Abbildung 2.12: Aufbau und Funktionsweise von A-GPS. [7]

Unter Android wird die Tracking- und LBS-Funktionalität vom Objekt `LocationManager` (siehe Abschnitt 2.1.6) bereitgestellt. Die gewünschte Methodik (radiolocation - netz- oder gerätebasiert und/oder satellitenbasiert) kann entweder explizit angegeben oder durch Festlegung verschiedener Kriterien eingestellt werden.

2.2.2 Georeferenzierung und Geotagging

Georeferenzierungen sind sehr nützlich, um Zusammenhänge in der Realität besser zu erklären und dienen zur einfacheren Wissenserkundung. Für Menschen sind raumbezogenene Informationen (Adressen, Ortsangaben, etc.) meist leicht zu verstehen, weil sie fast immer in Form von menschlicher Sprache ausgedrückt werden. Genau dieser Sachverhalt jedoch macht es für Computer sehr schwer, Ortsangaben zu erkennen und zu verarbeiten. Daher ist es notwendig, Geoinformationen nicht nur maschinenlesbar, sondern auch maschinenverstehbar bereitzustellen.[3]

[1] [7] S. 4
[2] Vgl. [7] S. 1-4
[3] Vgl. [9]

Eine Art und Weise, diese Information einzubinden, ist das sogenannte „Geotagging“ bzw. „Geocoding“. Es bezeichnet die Zuordnung von Koordinaten zu einem beliebigen Datensatz. Ein derartiger Datensatz kann ein Foto, Video, Dokument, eine Webseite bzw. im Grunde alles sein, was einen räumlichen Bezug hat. Diese Datensätze lassen sich sodann sehr leicht und automatisch in digitalen Kartensystemen an der referenzierten Position einbinden und darstellen. Für das Verorten von Fotos werden im Internet auch die Begriffe „Geo-Imaging“ und „Foto-Verortung“ verwendet.[1]

Anmerkung: Im Android SDK wird das Objekt `Geocoder` dazu verwendet, Koordinaten in Adressen umzuwandeln - und umgekehrt. Er hat somit nicht direkt mit der in dieser Sektion gemeinten Georeferenzierung von Datensätzen zu tun (siehe Abschnitt 2.1.6).

2.3 Akzelerometer

Ein Akzelerometer (englisch *accelerometer*) oder Beschleunigungssensor misst, wie der Name sagt, Beschleunigung. Beschleunigung definiert sich als die Änderungsrate der Geschwindigkeit. Somit detektiert ein derartiger Sensor, wie schnell sich seine Geschwindigkeit in einem gewissen Zeitabschnitt in Bezug zur Umgebung ändert. Damit kann einerseits simpel Bewegung festgestellt oder etwas komplexer (mit der zweiten Ableitung) die Geschwindigkeit berechnet werden.

Aufgrund physikalischer Gegebenheiten bezüglich der Messart eines Beschleunigungssensors ist Folgendes zu beachten:

> *„Accelerometers are unable to differentiate between acceleration due to movement and gravity. As a result, an accelerometer detecting acceleration on the Z-axis (up/down) will read -9.8m/s² when it's at rest.“*[2]

Anhand dieser Sensorwerte ist es neben der reinen Beschleunigungsmessung bzw. Geschwindigkeits*berechnung* auch möglich, die aktuelle Orientierung (Lage) des Geräts zu erkennen, beispielsweise ob das Gerät flach auf einer Oberfläche liegt, oder hochkant gehalten wird.

Beschleunigungen können in den drei verschiedenen Dimensionen (vorwärts/rückwärts, links/rechts und auf/ab) gemessen werden. Unter Android

[1] Vgl. [34]
[2] [3] S. 324

werden diese Daten - wie alle anderen Sensoren auch - über den `SensorManager` (siehe Abschnitt 2.1.6) bereitgestellt.[1]

Heutzutage werden 3D-Akzelerometer als Basis für Kontextsensitivität (englisch *context-awareness*) in vielen Geräten eingesetzt. Damit kann ein Gerät Daten aus seiner Umgebung (seinem Kontext) nutzen und daran sein Verhalten anpassen. Exemplarisch sei die Steuerung der Spielekonsole *Nintendo Wii* angeführt, wo man z.B. Tennis spielen kann, indem man den Controller ähnlich dem echten Tennisspiel vor- und zurückbewegt.

2.3.1 Schrittzähler

Ein Akzelerometer kann neben dem oben erwähnten Einsatz zur Steuerung eines Geräts auch dafür eingesetzt werden, „höhere physische Aktivitäten zu charakterisieren“, welche gerade in mobilen Anwendungsfällen eine Vielfalt an Kontextinformationen darstellen können. Damit kann ein Akzelerometer auch als Sensor zur Detektion von Schritten eingesetzt werden. Experimentell wurde herausgefunden, dass die kontinuierliche Auf- und Abbewegung des Geräts während der Schritte die besten Ergebnisse liefert. Da der Sensor die Daten abhängig von seiner Lage im Raum an den drei Achsen misst, ist es nicht möglich, einfach nur eine bestimmte Achse der Bewegung zuzuordnen.[2] Meine Vorgehensweise ist es daher, zuerst die Gerätelage zu bestimmen und die der Lage zugehörende Achse des Akzelerometers als Messbasis zu verwenden. Danach erfolgt die eigentliche Detektion eines Schritts:

1. Die Differenz zwischen aktuellem und letztem Messwert wird gebildet und einer Grenzwertprüfung unterzogen. Damit ist es möglich, etwaiges Sensorrauschen zu eliminieren.

2. Es wird überprüft, ob der vorherige Messwert im Ansteigen war (vorheriger Messwert ist größer als vorvorheriger Messwert) und der aktuelle Wert bereits im Sinken ist (aktueller Messwert ist kleiner als vorheriger Messwert). Ist dies der Fall, so wird ein Schritt detektiert.

Die Messdaten des Akzelerometers werden in Abbildung 2.13 veranschaulicht. Da ein Schritt einer periodischen Abfolge von Beschleunigungen entspricht, genügt es lediglich die Beschleunigungsänderung (entweder von positiv zu negativ oder von negativ zu positiv) zu detektieren.

[1] Vgl. [3] S. 323f
[2] Vgl. [6]

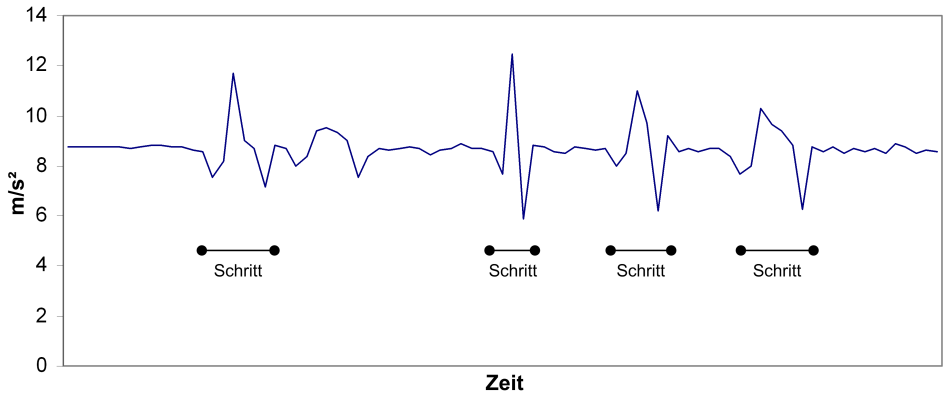

Abbildung 2.13: Messdaten des Akzelerometers mit detektierten Schritten.

2.4 Kommunikation

2.4.1 Datenanbindung

Waren früher die verschiedenen Mediendienstleistungen an ein fix zugeteiltes Verteilungsnetz geknüpft (z.B. Telefon und Telefonnetz, TV und Kabelnetz bzw. terrestrischer Empfang, ...) so kann man erkennen, dass mit den voranschreitenden Entwicklungen in der Digitaltechnologie und Signalverarbeitung diese Grenzen fallen. So werden immer mehr Leitungsnetze auf „Internet Protocol (IP) basierte packet-vermitteltet Technologien“ umgestellt. Beispiele hierfür sind u.a. VoIP (Voice over IP) und IPTV (IP Television). Da damit die Vermittlungsart vom konventionellen Trägermedium entkoppelt wird, ist es sehr leicht möglich, diese (nunmehr) *Services* über jegliche Netzwerke bereitzustellen, die IP-Fähigkeit aufweisen. Dies können verschiedene Funk-, als auch Festnetzleitungen sein.

Gleichzeitig mit dem Loslösen von den ursprünglichen fixen Kommunikationslösungen hin zu IP-basierter Leitungsvermittlung wurden auch immer mehr mobile Endgeräte mit immer mehr Anwendungen und mehreren (alternativen) Zugriffstechnologien ausgestattet. So sind viele Geräte mit Hardware ausgestattet, die es ermöglicht Datenkommunikation über Mobiltelefonnetze, WLan, u.a. durchzuführen.

Die Verwaltung aller Verbindungen, sodass der Benutzer bezüglich Access-

Netzwerk immer bestens bedient wird, ist das Feld aktueller Forschung.[1]

Unter Android ist für die Verwaltung der Kommunikations- und Datenverbindungen der `ConnectivityManager` (siehe Abschnitt 2.1.6) zuständig.

2.4.2 SMS

Das *Short Message Service* (SMS) ist eine Technologie, die von Mobiltelefonnetzen bereitgestellt wird. Es ermöglicht den Austausch von kurzen Nachrichten untereinander bzw. mit diversen Diensten. Obwohl SMS als Datenkanal im Vergleich zu anderen Technologien (siehe Sektion Datenanbindung) minderwertiger ist, gibt es durchaus auch viele Anwendungsbereiche, wo eine fixe Nachrichtenlänge (im ursprünglichen Standard 140 bytes) gepaart mit einer hohen Latenz, einer signifikant langsameren Datenrate und einer gewissen Verlustrate, trotzdem ausreichen.[2]

Um SMS unter Android zu verwenden steht der `SmsManager` (siehe Abschnitt 2.1.6) zur Verfügung. Seine Verwendung zeigt das Quellcodebeispiel in Listing 2.4.

```
1  // get sms manager
2  SmsManager sms = SmsManager.getDefault();
3  // define recipient and message
4  String to = "01234567";
5  String outStr = "hello world";
6  // define an intent to be fired when sms was sent
7  Intent intent = new Intent("SMS-SENT);
8  intent.putExtra("recipient", to);
9  PendingIntent sentIntent =
10     PendingIntent.getBroadcast(context, 0, intent, 0);
11 // send sms
12 sms.sendTextMessage(to, null, outStr, sentIntent, null);
```

Listing 2.4: Vorgang zum Versenden von SMS.

2.4.3 Bluetooth

Bluetooth ist eine Funktechnologie, um Geräte (z.B. Mobiltelefone, Headsets, Computer, ...) miteinander ohne Kabel zu verbinden. Es kommuniziert am 2.4 Ghz-Band und ist standardmäßig mit anderen Bluetoothgeräten kompatibel. Daten werden unter Verwendung von standardisierten Profilen mit anderen Geräten ausgetauscht.[3]

[1] Vgl. [10]
[2] Vgl. [11]
[3] Vgl. [12]

Unter Android sind bisher (Version 1.6) nur die Profile zum Anschluss von Kopfhörern und Headsets (A2DP und SCO) implementiert. Ein voll programmierbarer Bluetooth-Stack soll in Version 2.0 eingeführt werden[1](siehe auch Abschnitt 2.1.6).

2.5 Panoramafotos

Panoramafotos sind Fotos mit erweitertem Blickfeld. Sie bilden normalerweise keinen Moment, sondern eine Szene ab, wodurch derartige Bilder meist dynamischer wirken.[2] Um Panoramafotos zu erstellen, wurden viele Techniken entwickelt. Es gibt eigens dafür ausgelegte Kameras, weiters die Möglichkeit, das Blickfeld mit Hilfe von Fischauge-Objektiven zu vergrößern und Methoden, die Parabolspiegel zur direkten Aufnahme nutzen. Ein anderer Ansatz ist, eine Szene mit vielen aufeinanderfolgenden Einzelfotos zu fotographieren und anschließend mit geeigneten Algorithmen zu einem Ganzen zu verweben. Derartige Algorithmen werden als „Stitching"- bzw. „Image Mosaic"-Algorithmen bezeichnet.[3] Den letztgenannten Ansatz verfolge auch ich, um Panoramafotos zu erstellen. Mein Vorgehen lässt sich grob in zwei Teile einteilen: Akquirierung der Ausgangsbilder und Bildverarbeitung.

2.5.1 Akquirierung der Ausgangsbilder

Um die Ausgangsbilder zu erhalten gibt es wiederum zwei Möglichkeiten. Die erste Möglichkeit ist, viele Einzelbilder einer Szene nacheinander aufzunehmen. Dabei sollten sich die Bilder um ca. 50% überlappen.[4] Die zweite Möglichkeit ist, die Szene in einem Videoclip festzuhalten und anschließend ausgewählte Einzelbilder dem Bildverarbeitungsprozess zuzuführen.

Weil das Zielgerät einen Orientierungssensor integriert hat, mit dem der Zugriff auf Kompassdaten des Geräts möglich ist, möchte ich auch diese Kontextdaten nutzen. Dazu verwende ich die Kompassdaten um im Falle videobasierter Aufnahmeverfahren zu jedem Frame den aktuellen Richtungswinkel zu speichern (siehe Abbildung 2.14). Das hilft anschließend, den optischen Fluss in zwei aufeinanderfolgenden Frames grob abzuschätzen und damit die Suchregion einzuschätzen. Sam-

[1] Vgl. [26]
[2] Vgl. [13]
[3] Vgl. [14]
[4] Vgl. [35]

melt man die Ausgangsfotos mittels einzelner Fotos kann auch hier der Kompass helfen: Analog zu einem einrastenden Stativkopf kann er dazu verwendet werden, automatisch in eingestellten Intervallen zu fotographieren.

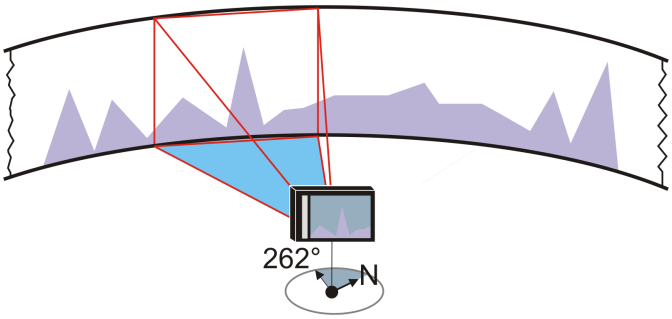

Abbildung 2.14: Verwendung des Kompasses zur Erstellung der Ausgangsfotos auf Basis der Blickrichtung.

2.5.2 Bildverarbeitung

Um die Einzelbilder zu einem Panoramabild zusammenzufügen, sind zwei Schritte notwendig: Das Finden von zusammengehörenden Regionen in jeweils aufeinanderfolgenden Einzelbildern und das Zusammensetzen der Fotos. Dabei wird auf Basis dieser Referenzen das Panoramafoto aufgebaut.

Finden von Referenzpunkten

Zuerst müssen gemeinsame Punkte in aufeinanderfolgenden Einzelbildern gefunden werden. Mein Ansatz für diese Aufgabe basiert auf der blockbasierten Bewegungsabschätzung.[1] Dazu wird jedes Bild in Blöcke von je 8x8 Pixel unterteilt und jedem Block der Durchschnittswert der 64 Farbwerte zugewiesen. Daraufhin werden im Bild, für das die Verschiebung berechnet wird, 2x2 Blöcke bestimmt, die im vorherigen Bild gesucht werden sollen. Ich verwende dabei die Blöcke an den Positionen $(0, [Bildhöhe/2] - 1)$ bis $(1, [Bildhöhe/2])$. Unter der Annahme, dass die Einzelbilder durch einen Schwenk von links nach rechts aufgenommen wurden und sich um ca. 50% überdecken, kann davon ausgegangen werden, dass diese Bildregion auch im vorherigen Einzelbild vorkommt (siehe Schritt 1 in Abbildung 2.15).

[1] Vgl. [13]

Als nächstes wird diese Region im anderen Einzelbild gesucht (Abbildung 2.15, Schritt 2). Dazu wird ein Intervall festgelegt, in dem der Farbwert zweier Blöcke als gleich erachtet wird. Weil dies für mehrere Blöcke zutreffen kann, ist das Ergebnis dieses Vorgangs eine Liste an möglichen Referenzkandidaten. Aus diesen Kandidaten wird daraufhin der beste Kandidat als zur Suchregion passend bestimmt (Abbildung 2.15, Schritt 3). Der beste Kandidat ist dabei definiert durch die kleinste Farbabweichung vom Suchblock. Das Endergebnis der Suche ist ein Referenzvektor, der die Verschiebung von zwei aufeinanderfolgenden Einzelbildern angibt. Dieser Vektor bildet die Basis für das abschließende Zusammensetzen der Fotos.

Der Hauptgrund, nach nur einer 2x2-Region zu suchen, ist, dass eine aufwändigere Suche nach Referenzregionen direkt in einer höheren Rechenzeit und einem höheren Speicherverbrauch resultieren würde. Angenommen man würde Referenzen aller Blöcke des Bilds im vorherigen Bild suchen, um daraus den Referenzvektor als Durchschnittswert zu berechnen, ergäbe sich ein grobes Laufzeitverhalten der Ordnung $O(n^4)$. Mein Suchalgorithmus hat die Ordnung $O(n^2)$. Da mobile Endgeräte gerade in Bezug auf CPU- und Speicherkapazitäten sehr beschränkt sind, ist für eine erhöhte Performanz ein derartiger Algorithmus vorzuziehen.

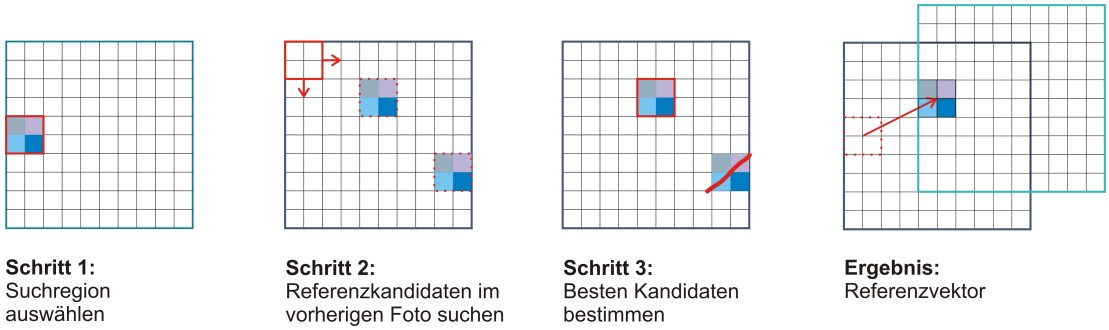

Abbildung 2.15: Vorgang zur Suche nach Referenzpunkten.

Zusammensetzen der Fotos

Auf Basis der im vorigen Schritt gewonnenen Vektoren wird nun das Panoramafoto erstellt. Um den visuellen Eindruck zu verbessern, werden die einzelnen Einzelbilder

ineinander überblendet.[1]

2.6 Visualisierung

Um die gesammelten Daten (vornehmlich GPS-Koordinaten, Fotos, gezählte Schritte) zu visualisieren, wird *Google Maps* verwendet. Dazu steht unter Android das UI-Objekt *MapView* zur Verfügung. Gleichzeitig werden die Daten auch im persönlichen Google Maps-Konto im Internet gespeichert und können dort ebenfalls in gewohnter Weise aufgerufen und angezeigt werden.

2.6.1 Visualisierung unter Android

Das Pendant zu Googles Kartendienst *Google Maps* stellt unter Android das UI-Element *MapView* dar. Es bietet die Funktionalität, Karten anzuzeigen und eigene Informationen in Form von *Overlays* hinzuzufügen. Weiters kümmert es sich um die Transformation von Geokoordinaten in Bildschirmkoordinaten (und umgekehrt) und um das Steuern (Verschieben, Zooming, Bestimmen der Kartenart, u.a.) der Karte.

Um eine MapView zu verwenden, muss sie anstatt in einer Activity in einer `MapActivity` eingebettet werden. Dabei leistet die MapActivity die benötigte Hintergrundarbeit, eine MapView anzuzeigen, wie beispielsweise das adäquate Herunterladen der Kartenkacheln von den Google-Servern. Damit eine MapView funktioniert, muss zuvor ein Maps API Schlüssel von Google bezogen und eingetragen werden (siehe Abschnitt 2.1.7). Zur Illustration zeigt Abbildung 2.16 eine einfache MapActivity mit einer MapView.[2]

Anmerkung: Wie bereits früher (Abschnitt 2.1.2) erwähnt, ist Google Maps Teil der proprietären Google Apps und somit nicht standardmäßig in Android enthalten. Will man es nutzen, muss im Manifest auf das Package `com.google.android.maps` verwiesen werden.

[1] Vgl. [35]
[2] Vgl. [36]

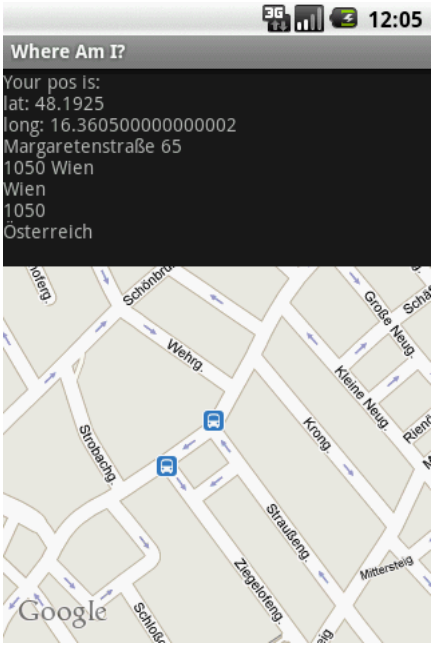

Abbildung 2.16: Anwendung *Where Am I?* mit einer MapView. [3] S. 229

2.6.2 Visualisierung im Internet

Um die am mobilen Gerät ermittelten Daten auch im Internet zugänglich zu machen, werden die Daten mittels Datenanbindung ins Web geladen und gespeichert. Dazu verwende ich die Plattformen *Google Maps* für Positionsdaten und *Picasa Web Albums* für Fotos. Sie bieten die Möglichkeit zur adäquaten Darstellung der Daten und zusätzlich verschiedene Einstellungen bezüglich der Privatsphäre, deren Schutz gerade bei Geodaten über den persönlichen Aufenthaltsort besonders wichtig ist.

Zugriff auf die Dienste gewährt Google durch die offen gelegten Programmierschnittstellen *Google Maps Data API* und die *Picasa Web Albums Data API*. Sie basieren beide auf dem *Google Data Protocol* und ermöglichen das Anmelden an den Google-Servern mit dem persönlichen Konto sowie das Einsehen und Manipulieren (Hinzufügen, Ändern, Löschen) der Daten.

Den schematischen Informationsfluss vom Gerät bis zur finalen Darstellung im Browser zeigt Abbildung 2.17: GPS-Daten, Bilder und gezählte Schritte werden im mobilen Endgerät ermittelt und über eine Internetanbindung auf den

Google-Servern gespeichert. Von dort können diese Daten unter Verwendung der herkömmlichen Google-Dienste (`maps.google.com` und `picasaweb.google.com`) in einem Internetbrowser visualisiert und wiederum auch weiterverarbeitet werden.

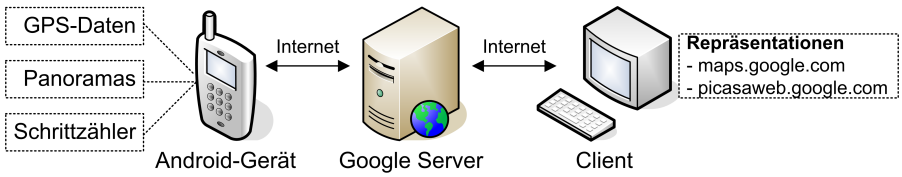

Abbildung 2.17: Schematischer Ablauf des Informationsflusses.

Die Basis der APIs bildet das *Google Data Protocol* (kurz *gData Protocol*), das die Art und Weise definiert, wie Daten ausgetauscht werden. Es stellt die Datensätze in Form von *Feeds* bereit und basiert auf dem Syndication-Format AtomPub.
Die grundlegende Idee des Protokolls ist, jedem Datensatz eine eindeutige URI zuzuweisen, auf welche mittels der HTTP-Requests GET, POST, PUT und DELETE (auch manipulativ) zugegriffen werden kann. Der syntaktische Aufbau der Feeds unterliegt dabei einem vorgegebenen XML-Schema, wodurch die Daten von jeder netzwerkfähigen Plattform (somit auch von einem Android-Gerät) geparst und weiterverwendet werden können.
Anmerkung: Um Daten aus dem persönlichen Google-Konto abrufen zu können bedarf es eines vorherigen Logins. Dabei wird ein *Authentification-Token* generiert, der jeder Anfrage hinzugefügt werden muss und dadurch den jeweiligen Benutzer definiert.[1]

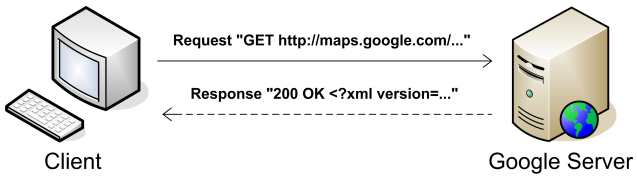

Abbildung 2.18: Ablauf einer gData-Kommunikation.

Abbildung 2.18 verdeutlicht den grundlegenden Aufbau des Protokolls: Eine HTTP GET-Anfrage wird vom Client zum Google-Server gesendet

[1] Vgl. [37]

```
GET http://maps.google.com/maps/feeds/maps/userID/full/mapID
Authorization: GoogleLogin auth='authorizationToken'
```

Die URI definiert dabei:

- das Transportprotokoll: `http://`
- den Server: `maps.google.com`
- das Service: `/maps` (hier das Kartenservice)
- die Antwortart: `/feeds` (hier als AtomPub-Feed)
- den User: `/userID`
- den konkreten Eintrag: `/full/mapID` (hier die Karte `mapID`)

Es wird also die Karte `mapID` vom Benutzer `userID` als AtomPub-Feed vom Server `maps.google.com` angefordert. Der Server antwortet daraufhin mit der XML-Beschreibung der konkreten Karte:

```
200 OK
<?xml version='1.0' encoding='utf-8'?>
<entry xmlns='http://www.w3.org/2005/Atom'>[...]</entry>
```

Die HTTP-Anfragen `POST` (hinzufügen), `PUT` (ändern) und `DELETE` (löschen) funktionieren nach dem analogen Schema. Der Statuscode (in diesem Beispiel `200 OK`) gibt Auskunft über den Erfolg oder Misserfolg der aktuellen Anfrage.[1]

Die *Google Maps Data API* und die *Picasa Web Albums API* bedienen sich beide dem Anfrage-Antwort-Prinzip des Protokolls, implementieren jedoch verschiedene XML-Schemas, um die verschiedenen Datenarten (Karten und Fotos) mit ihren speziellen Eigenschaften (z.B. Metatags eines Fotos) innerhalb des `<entry>`-Tags abzubilden.[2]

[1] Vgl. [38]
[2] Vgl. [39]

Kapitel 3

Projektbeschreibung und Modellierung

Dieses Kapitel befasst sich mit der Modellierung der praktischen Anwendung. Ausgehend von den Anforderungen an das Programm werden Anwendungsfälle erstellt, welche wiederum die Basis für die Aktivitätsdiagramme sind. Aus den Aktivitätsdiagrammen werden sodann die Klassen abgeleitet und in Klassendiagrammen dargestellt. Das Ziel dieses Vorgangs ist, die wichtigsten Funktionalitäten besser verstehen und eine visuelle Einsicht in die Programmarchitektur geben zu können. Aufbauend auf diesen Modellen wird das eigentliche Programm implementiert. Die Notation der Diagramme ist UML-konform.

3.1 Anforderungsanalyse

Ausgehend vom in Kapitel 1 angeführten Projektziel ergeben sich folgende Anforderungen an die fertige Anwendung:

- Der Standort des Geräts soll fortlaufend, in einstellbaren Intervallen ermittelt und aufgezeichnet werden. Der Standort definiert sich als Längen- und Breitengrad sowie der Höhenangabe. Anhand der zusätzlichen Höheninformation kann im Visualisierungsschritt ein Höhenprofil generiert werden.
- Mithilfe des Akzelerometers soll ein Schrittzähler erstellt werden. Dieser zählt einerseits die getanen Schritte und schätzt andererseits davon ausgehend die Bewegungsart (Laufen, Gehen, Stehen, Stopp) ab.

- Wird vom Schrittzähler eine gewisse Zeit kein Schritt detektiert so wird eine Verunglückung angenommen und es werden Rettungsmaßnahmen ergriffen. Diese sind das Ausnutzen aller vorhandener Kommunikationskanäle um Notfallmeldungen inklusive genauer Ortsangabe an vorher definierte Personen abzusetzen. Genutzt werden die Kanäle Mobilnetz, Bluetooth und WLan. Von anderen Teilnehmern empfangene Ortsangaben werden gespeichert und können visualisiert werden

- Es soll die Möglichkeit geben, den Standort eines anderen Teilnehmers anzufragen. Das Gerät des Teilnehmers sendet den aktuellen Standort selbsttätig an den Anfragenden zurück. Auch diese Ortsangabe wird gespeichert und bei Bedarf visualisiert.

- Panoramas werden durch das Schwenken der Kamera über die Landschaft automatisch erstellt. Sie werden zusammen mit der zugehörigen Georeferenz gespeichert und ebenfalls bei Bedarf visualisiert.

- Die erzeugten Daten werden am Gerät in einer MapView und im Web im persönlichen Google Maps Konto visualiert. Die Datenübertragung erfolgt, eine mobile Internetanbindung vorausgesetzt, in Echtzeit.

Technische Anforderungen

Zur Anwendungsentwicklung werden nur das Java Development Kit (JDK) und das Android Software Development Kit (Android SDK) benötigt. Zur Vereinfachung der Entwicklung gibt es jedoch zusätzlich das Android Development Tools (ADT) -Plugin für die Eclipse IDE (siehe Abschnitt 2.1.7).

Zum Betrieb und zum Testen des Programms in einem Real-Life-Szenario ist ein Android-Mobiltelefon mit Unterstützung von GPS- und Beschleunigungssensoren sowie einer mobilen Internetanbindung erforderlich. Zur Nutzung der Peer-to-Peer-Funktionalitäten werden entsprechend mehr Geräte benötigt. Der im Android SDK mitgelieferte Emulator reicht zwar aus, die Grundfunktionen zu testen. Da die Sensorwerte des Akzelerometers sowie der Kamera-Input derzeit nicht emuliert werden können, ist aber für dieses Projekt ein Testgerät unabdingbar.

3.2 Modellierung

3.2.1 Anwendungsfalldiagramm

Anhand der Anforderungsanalyse ergeben sich die Anwendungsfälle *tracking+analyzing*, *rescue system*, *analyze*, *track location*, *make panorama*, *visualize*, *request location*, *edit preferences*, *store data* und *send to google* (siehe Abbildung 3.1). Die Anwendungsfälle *tracking+analyzing*, *make panorama*, *visualize*, *request location* und *edit preferences* interagieren in irgendeiner Weise mit dem Benutzer. Zusätzlich schließt der Anwendungsfall *tracking+analyzing* das Rettungssystem (*rescue system*), das Schrittzählen und Höhenanalysieren (*analyze*) und das eigentliche Positionsbestimmen (*track location*) ein. Der Anwendungsfall *store data* ist für die Verwaltung der erzeugten Daten zuständig und wird auch von allen anderen Anwendungsfällen verwendet. Zur Verbesserung der Übersicht wurde dieser Sachverhalt im Diagramm jedoch weggelassen. Der *store data*-Anwendungsfall wird automatisch vom System betreut und damit einem Admin zugeordnet. Weiters wird er vom Anwendungsfall *send to google* erweitert, der die Übertragung der Daten an Google behandelt.

3.2.2 Aktivitätsdiagramme

Anhand der oben angeführten Anwendungsfälle ergeben sich Funktionalitäten, deren Abläufe in den folgenden Aktivitätsdiagrammen dargestellt werden. Grau hinterlegte Diagramme weisen auf aggregierte Aktionen hin. Für sie finden sich in Folge detaillierte Aktivitätsdiagramme.

Die inneren Abläufe des Anwendungsfalls *track+analyze* werden in Abbildung 3.2 gezeigt. Diese Funktionalität wird nunmehr *Trackalyzer*, ein Kunstwort aus *Tracker* und *Analyzer*, genannt. Er bildet das zentrale funktionale Element des Programms und kümmert sich, sobald gestartet, um das fortwährende Schrittzählen, Orten und Eventhandling. Auch das Absetzen von Notfallmeldungen (Aktion *alert others*) wird von ihm erledigt, was in Abbildung 3.5 genauer darstellt ist.

Die Aktion *sms location request receiver* (Abbildung 3.7) behandelt Anfragen, wenn andere die Position mittels der Aktion *sms location requester* (Abbildung 3.6) abfragen. Auf diese Anfrage antwortet der *sms location sender* (Abbildung 3.8), worauf im empfangenden Gerät wiederum der *sms location respond receiver* (Abbildung 3.9) reagiert. Die Aktion *sms location sender* wird auch zum selbsttätigen Absetzen einer Notfallmeldung verwendet. Diese SMS-Kommunikationsabfolge wird

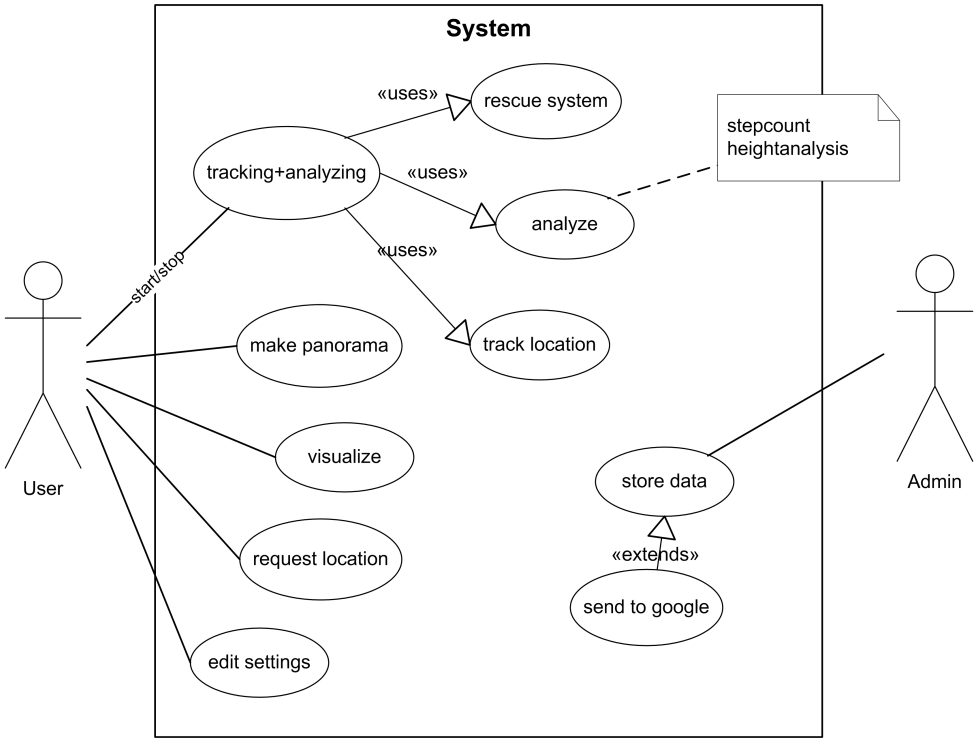

Abbildung 3.1: Anwendungsfalldiagramm des Projekts.

auch im Sequenzdiagramm in Abbildung 3.13 dargestellt.

Weiters zeigt *edit settings* (Abbildung 3.10) die Umsetzung zum Verändern der Voreinstellungen, *make panorama* (Abbildung 3.11) die notwendigen Aktionen zum Erstellen der Panoramafotos und *visualize* die Abfolge um eine ausgewählte Trackalyzer-Route mitsamt ihrer Log-Einträge in einer Karte zu visualisieren.

Mit der Aktion *announce feature* (siehe Abbildung 3.3) und ihrer zugehörigen Aktion *feature receiver* (siehe Abbildung 3.4) werden die so genannten *Features* gespeichert. Dieser Begriff ist der Terminologie der *Web Map Services* (WMS) entlehnt und charakterisiert dabei eine Entität, die in eine Karte eingezeichnet wird. In diesem Projekt sind das die verschiedenen Koordinaten einer aufgezeichneten Route sowie auch die georeferenzierten Panoramafotos. Durch den Aufbau mit einem *announcer* und *receiver* wird zusätzlich eine androidspezifische Methodik eingesetzt: Man kündigt das Vorhandensein neuer Features mittels eines Broadcast Intents (Abschnitt 2.1.5) systemweit an, fängt diese Meldung mit einem zugehörigen Receiver wieder ab und reagiert darin in gewünschter Weise. In diesem Fall wird das Feature vom Objekt `FeatureReceiver` empfangen, in der `TrackalyzerDB` ge-

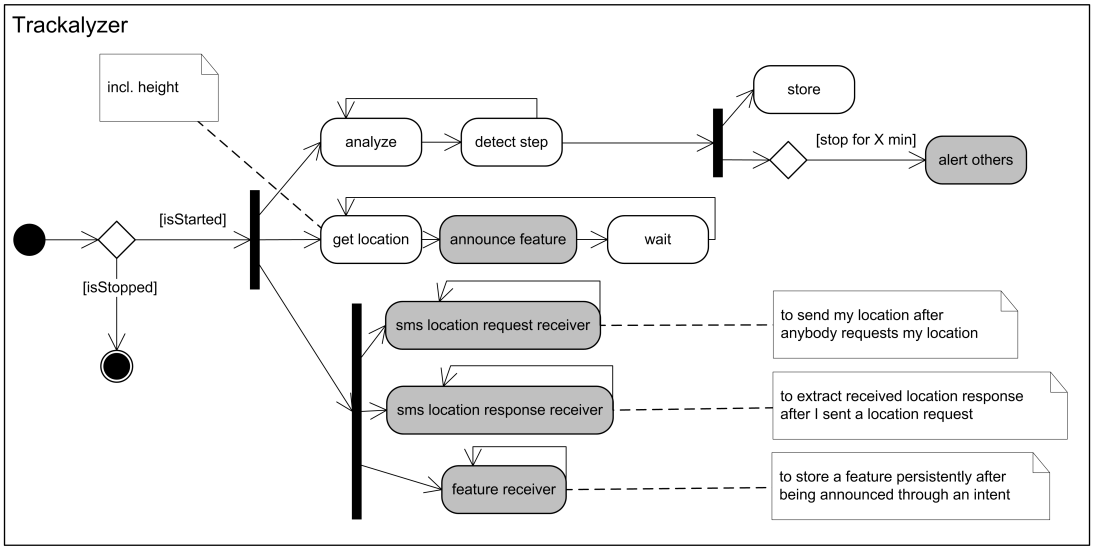

Abbildung 3.2: Aktivitätsdiagramm *Trackalyzer*.

speichert und ins Internet weiterübertragen. Zur Illustration dieses Vorgangs sei auch auf das Sequenzdiagramm in Abbildung 3.14 verwiesen. Diese Funktionalität durch das Intent-System zu realisieren bringt einige Vorteile: Erstens wird damit die Panoramafunktion vom übrigen (Trackalyzer-)Programm entkoppelt, zweitens gibt man damit auch Fremdentwicklern die Chance, auf neue Features zu reagieren und drittens könnte damit auch ein Fremdentwickler die Infrastruktur (Speichern von Kartenfeatures) dieses Programms nützen indem er in seinem Programm einen passenden Intent absetzt. Man denke hierbei zum Beispiel an ein Programm, das anstatt Panoramas simple Fotos schießt und dies mit einem adäquaten Intent dem System mitteilt. Die Trackalyzer-Anwendung würde daraufhin das Foto in analoger Weise wie ein Panorama behandeln, es also in ihrer Datenbank speichern bzw. genauso auch ins Web laden.

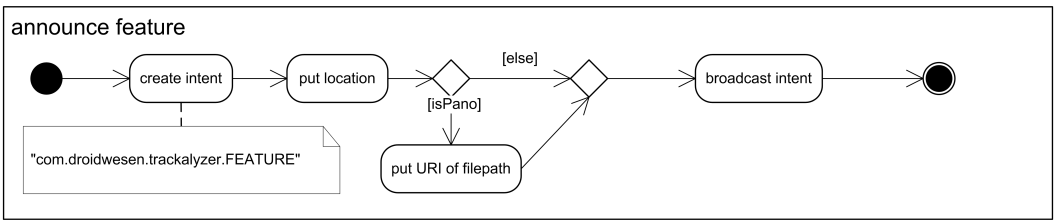

Abbildung 3.3: Aktivitätsdiagramm *announce feature*.

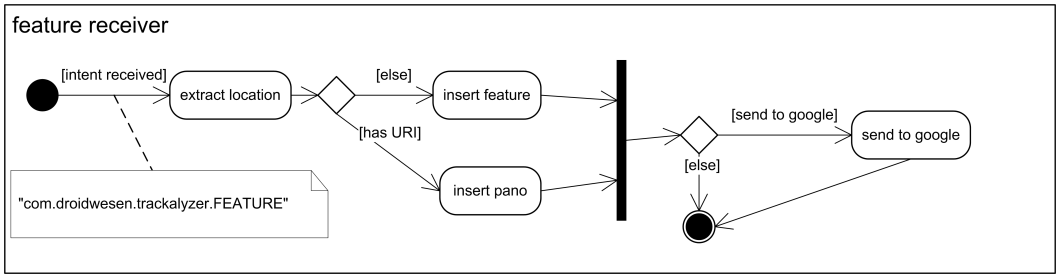

Abbildung 3.4: Aktivitätsdiagramm *feature receiver*.

Anhand des Aktivitätsdiagramms *alert others* (siehe Abbildung 3.5) ist das Alarmierungssystem ersichtlich. Es wird der aktuelle Standort ermittelt und per SMS übertragen. Zusätzlich werden die persönlichen Funknetze Bluetooth und WLan (WiFi) aktiviert und es wird kontinuierlich nach Geräten in Reichweite gesucht. Nach Entdeckung eines Geräts wird diesem der Standort übermittelt.

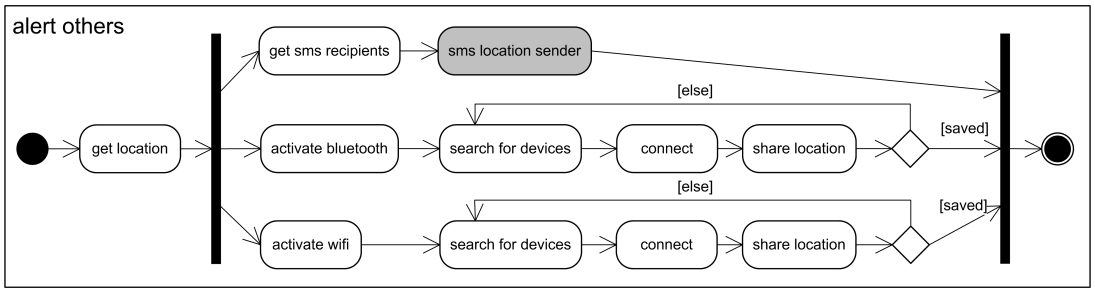

Abbildung 3.5: Aktivitätsdiagramm *alert others*.

Die Aktion *sms location requester* (siehe Abbildung 3.6) beschreibt den Vorgang, die Position einer anderen Person anzufordern. Es wird eine SMS mit dem Inhalt `where?` erstellt und an den Empfänger gesendet. Außerdem wird dies in der Log-Datenbank protokolliert.

Das Aktivitätsdiagramm *sms location request receiver* in Abbildung 3.7 zeigt den Verlauf, wenn eine Standort-Anfrage empfangen wird. Zuerst wird überprüft, ob die SMS das Wort `where?` beinhaltet und ob sie von einer zulässigen Rufnummer gesendet worden ist. Ist beides der Fall, wird der aktuelle Standort ermittelt und dieser an den Anfragenden mit der Aktion *sms location sender* (Abbildung 3.8) zurück gesendet. Auch dieser Vorgang wird in der Log-Datenbank vermerkt.

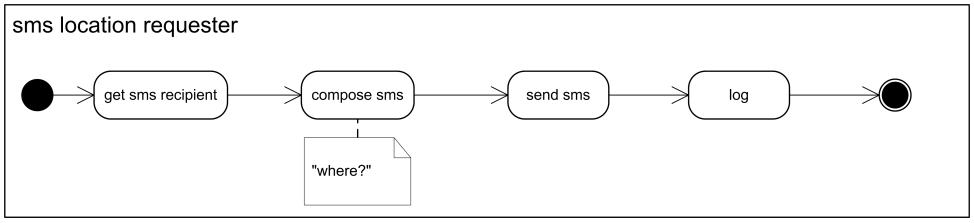

Abbildung 3.6: Aktivitätsdiagramm *sms location requester*.

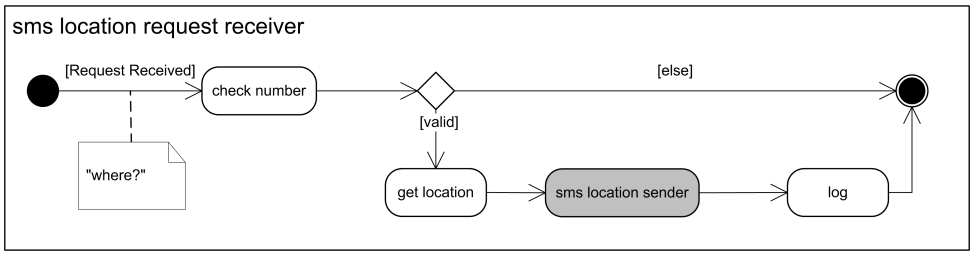

Abbildung 3.7: Aktivitätsdiagramm *sms location request receiver*.

Um einen Standort mittels SMS zu versenden wird die Aktion *sms location sender* verwendet. Ihr Aktivitätsdiagramm zeigt Abbildung 3.8. Eine SMS im Format `iam@(LAT/LNG)` wird erstellt, an den zuvor ermittelten Empfänger gesendet und der Vorgang ebenso protokolliert. Die Koordinaten `(LAT/LNG)` stellen den Standort in Längen- und Breitengraden dar.

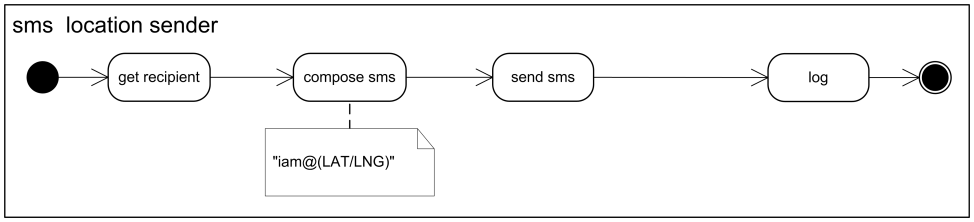

Abbildung 3.8: Aktivitätsdiagramm *sms location sender*.

Das Aktivitätsdiagramm in Abbildung 3.9 zeigt die Aktion *sms location respond receiver*. Sie wird ausgeführt, wenn ein Standort empfangen wird. Beinhaltet die SMS die Zeichenkette `iam@(LAT/LNG)` und stammt sie von einer zulässigen Rufnummer, so werden die Koordinaten geparst und im Protokoll eingetragen. Danach wird der Benutzer über diesen Vorgang informiert.

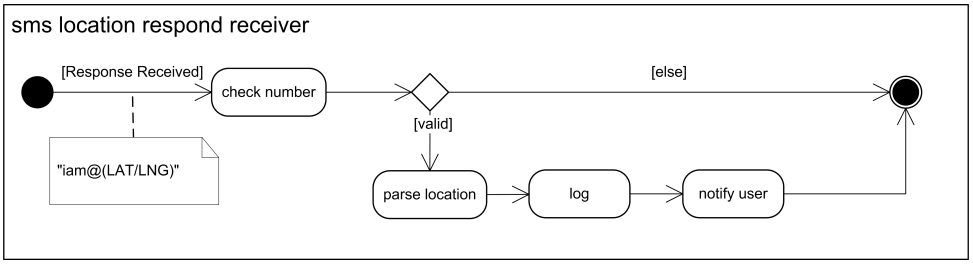

Abbildung 3.9: Aktivitätsdiagramm *sms location respond receiver*.

Mithilfe der Aktion *edit preferences* (siehe Abbildung 3.10) werden die Voreinstellungen definiert. Diese Einstellungen betreffen die Tracking-Intervalle und das automatische Übertragen der Daten an Google. Auch die Google-Login-Daten werden hier festgelegt.

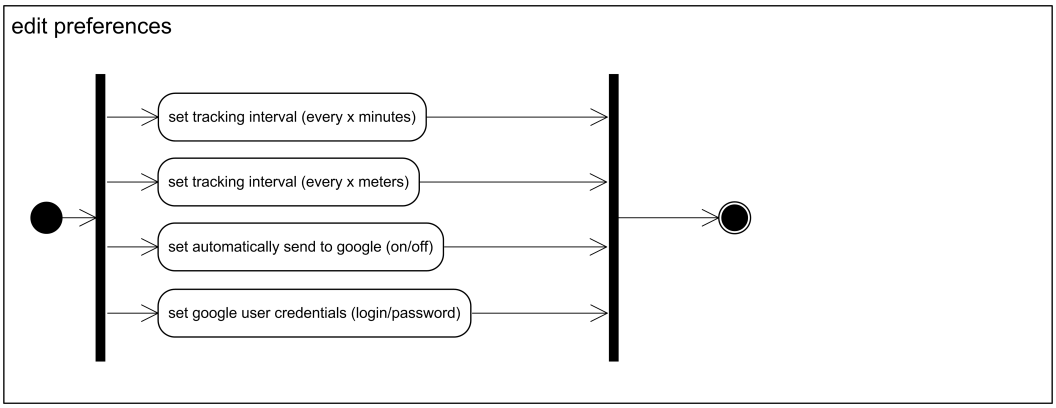

Abbildung 3.10: Aktivitätsdiagramm *edit preferences*.

Die Abfolge zum Erstellen von Panoramafotos zeigt Abbildung 3.11. Das Ausgangsmaterial wird gesammelt, indem die Kamera über das Blickfeld geschwenkt wird und die Bilddaten in einem Videoclip oder in mehreren Fotos gespeichert werden. Anschließend werden diese Daten zu einem Panoramafoto verarbeitet, dieses Foto auf der Speicherkarte gespeichert und mit der Aktion *announce feature* (siehe Abbildung 3.3) dem System angekündigt.

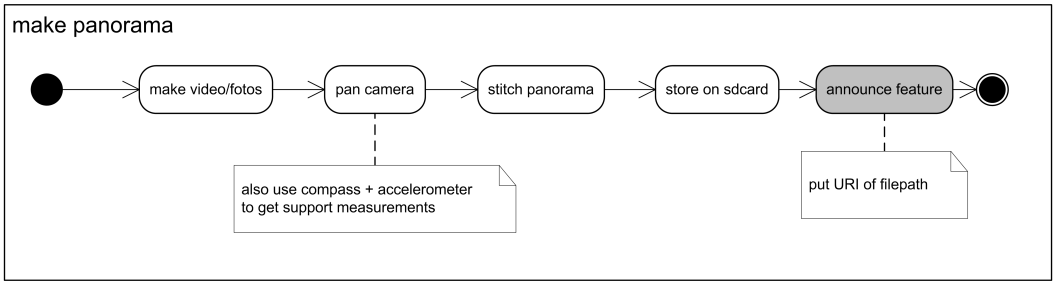

Abbildung 3.11: Aktivitätsdiagramm *make panorama*.

Das Aktivitätsdiagramm in Abbildung 3.12 zeigt den Vorgang zum Visualisieren der gewonnenen Daten. Nachdem die gewünschte Route ausgewählt wurde, werden die gespeicherten Features und Protokolldaten aus den Datenbanken geladen und in einer Landkarte an den zugehörigen Positionen eingezeichnet.

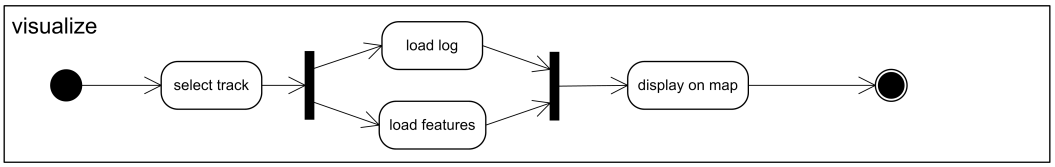

Abbildung 3.12: Aktivitätsdiagramm *visualize*.

3.2.3 Sequenzdiagramme

Zur Veranschaulichung von Interaktionen zwischen Objekten und ihrer zeitlichen Zuordnung stehen in der UML die Sequenzdiagramme zur Verfügung. Diese verwende ich in meinem Projekt zur Illustration der SMS-Kommunikation sowie zur Darstellung der Vorgänge *announce feature* und *receive feature*.

In Abbildung 3.13 wird die Funktionsweise der SMS-Kommunikation zwischen zwei Endgeräten dargestellt und auf die zuständigen Aktivitäten verwiesen.

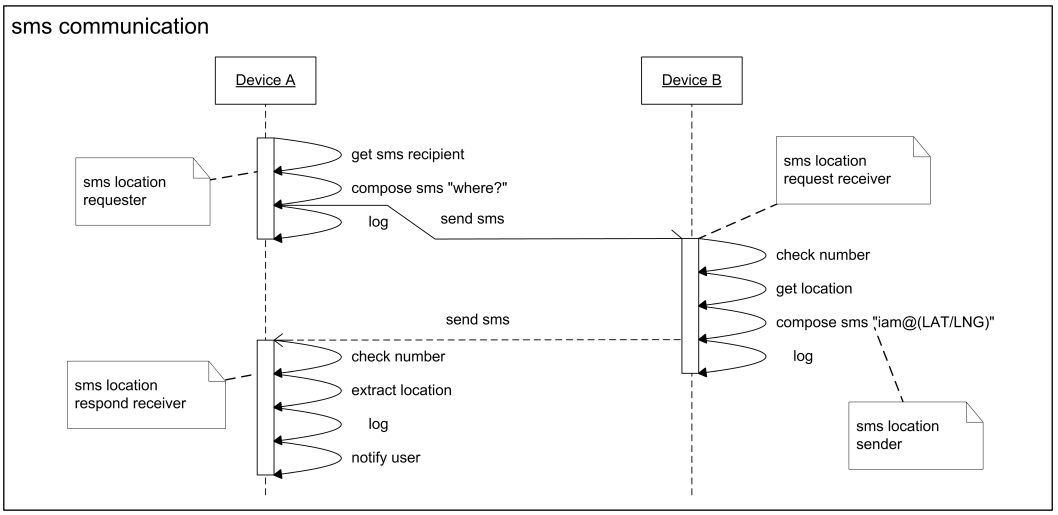

Abbildung 3.13: Sequenzdiagramm *sms communication*.

Abbildung 3.14 zeigt die durchzulaufenden Abfolgen, wenn ein Feature gespeichert wird. Zuerst wird der aktuelle Standort ermittelt, dieser in einen Intent verpackt und anschließend an das System gesendet. Der dafür zuständige `FeatureReceiver` im `Trackalyzer` nimmt sich dem Intent an und fügt die Informationen in der Datenbank ein (bzw. übermittelt sie an die Google-Server). Fast zeitgleich sendet auch der Panographer einen Intent an das System und kündig somit das Vorhandensein eines neuen Panoramafotos an. Dass der Trackalyzer auch diesen Intent behandelt wurde im Diagramm jedoch der Einfachheit halber nur mit drei Punkten angedeutet. In diesem Diagramm sind zusätzlich die Asynchronität und der eventgesteuerte Aspekt des Intent-Systems sehr gut ersichtlich.

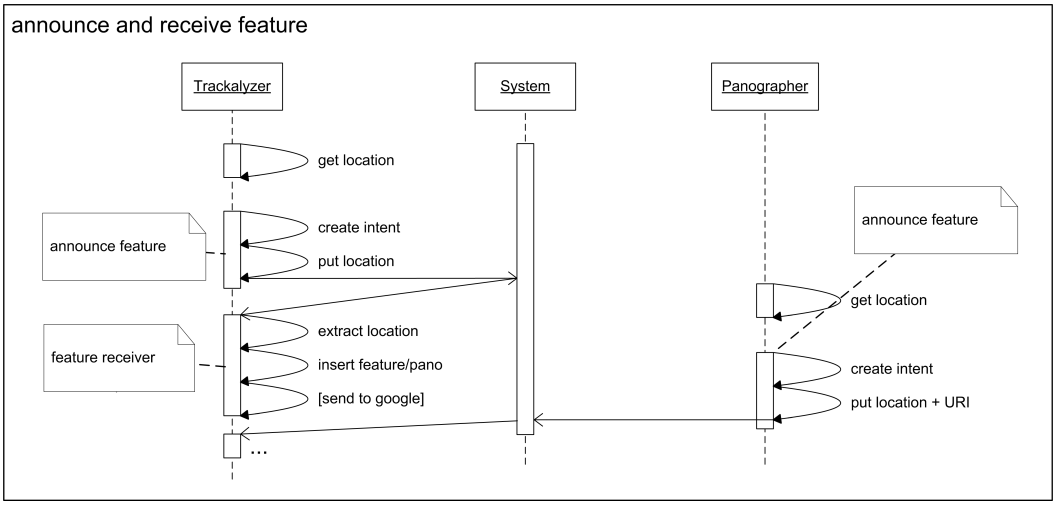

Abbildung 3.14: Sequenzdiagramm *announce and receive feature*.

3.2.4 Klassendiagramme

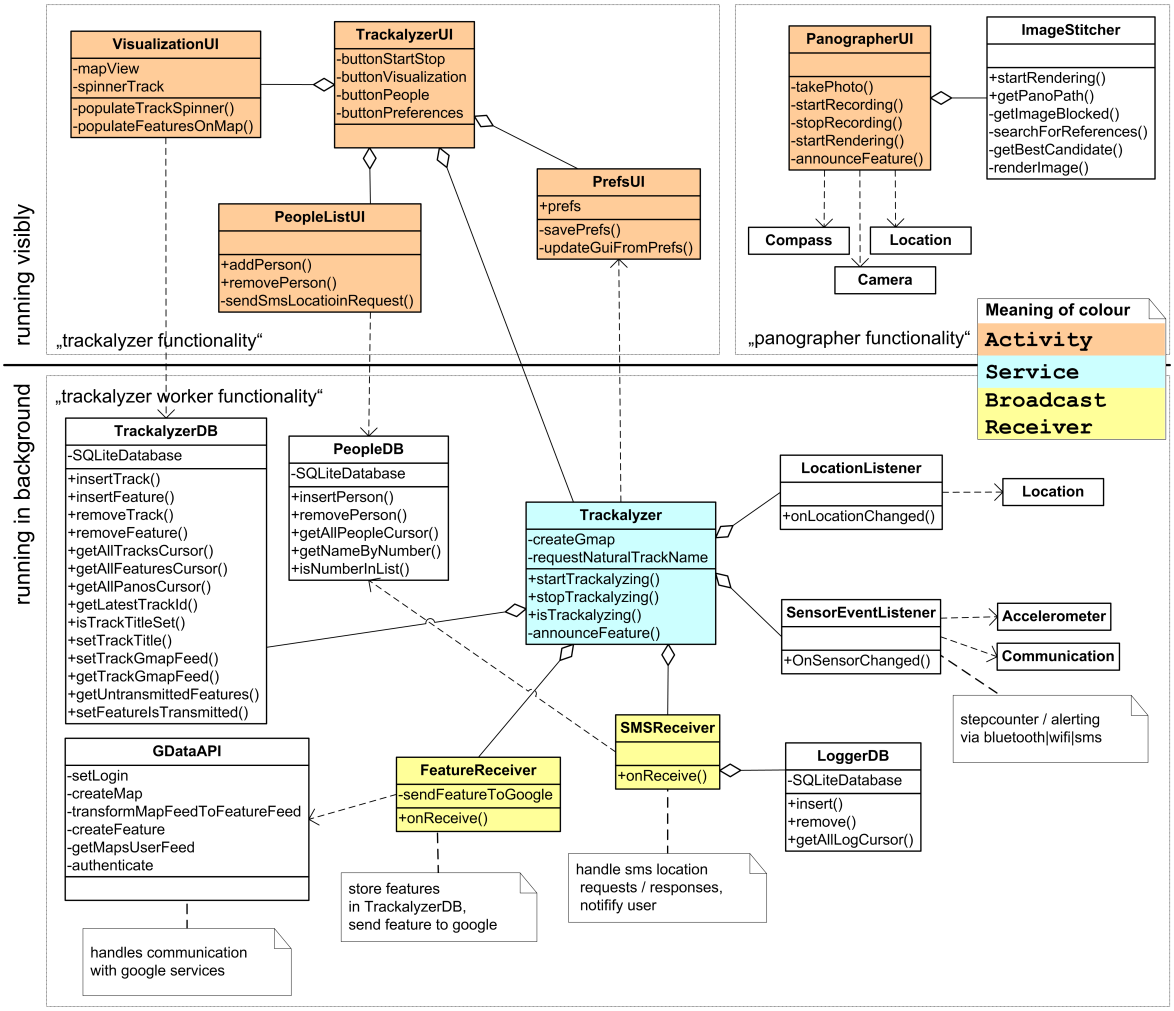

Abbildung 3.15: Gesamtübersicht über alle Klassen.

Aus all den vorangegangenen Überlegungen geht nun ein Gesamtklassendesign hervor, das in Abbildung 3.15 dargestellt ist. Es zeigt die verschiedenen Komponenten und deren Zusammenspiel. Weiters sind bereits die in Android zu verwendenden Schlüsselkonzepte (Abschnitt 2.1.5) miteingearbeitet. So stellen orange Klassen eine `Activity`, hellblaue Klassen ein `Service` und gelbe Klassen einen `BroadcastReceiver` dar. Die Objekte *Accelerometer*, *Camera*, *Communication*,

Compass und *Location* meinen Android-APIs, sind jedoch zur Vereinfachung des Modells dem Namen nach nicht mit ihnen ident.

Des Weiteren präsentiert sich in diesem Modell auch noch eine andere Sicht auf die Architektur des Projekts. So sammelt sich sämtliche *Worker-Funktionalität* innerhalb des `Trackalyzers`, der durch ein `Service` implementiert ist und damit per Definition unsichtbar ist und nur im Hintergrund läuft (siehe Abschnitt 2.1.4). Im Gegensatz dazu befinden sich oberhalb die `Activities`, welche per se eine Oberfläche haben und daher den sichtbaren Prozessen zugeordnet werden. Außerdem erkennt man die Entkopplung des `Panographers` vom übrigen Trackalyzer. Mittels der Methode `announceFeature()` bindet er sich jedoch in das Gesamtsystem ein (siehe Sequenzdiagramm announce and receive feature in Abbildung 3.14).

Anhand dieser Aufteilung bietet sich auch die Aufteilung in verschiedene Packages an: Ein Paket für *trackalyzer worker functionality* und *trackalyzer functionality* und ein zweites für die *panographer functionality*. Es ist auch möglich, den `Panographer` alleinstehend, also komplett ohne die *trackalyzer (worker) functionality* zu implementieren. Sein *announce feature*-Intent würde dann jedoch ungehört bleiben.

3.2.5 Datenbank-Schemas

Zur persistenten Datenspeicherung der erzeugten Daten wird das nativ in Android integrierte relationale DBMS SQLite (siehe Abschnitt 2.1.6) verwendet. Im Gesamtklassendiagramm sind die erforderlichen Datenbanken ersichtlich. Diese sind *TrackalizerDB*, *LoggerDB* und *PeopleDB*. Hier werden ihre Tabellen in gängigen Entity Relationship (ER) -Diagrammen dargestellt (siehe Abbildung 3.16).

TrackalyzerDB in Abbildung 3.16 definiert das Datenmodell, das der Speicherung der aufgezeichneten Tracks zugrunde liegt. Ein Track besteht aus mehreren Features, wobei ein Panoramafoto als Feature mit zusätzlich einer URI (Dateipfad zum Bild) modelliert wird. Die Eigenschaft `gmapFeed` in der Tabelle Track speichert die URI zur Google Map im Internet. Das Flag `onGoogle` in der Tabelle Feature kennzeichnet, ob ein Feature bereits erfolgreich in die Google Map übertragen worden ist. Damit wird eine Warteschlange realisiert, die besonders dann zum Tragen kommt, wenn ein Feature aufgrund einer instabilen Netzwerkanbindung nicht übertragen werden konnte. Dann wird versucht, das Feature beim nächsten Mal mit zu übertragen.

LoggerDB in Abbildung 3.16 stellt den Aufbau der Protokolldatenbank dar. Hierin werden die auftretenden Events der SMS-Kommunikation geloggt und *PeopleDB* in Abbildung 3.16 zeigt die Verwaltung der Personen, die im Notfall benachrichtigt werden. Gleichzeitig haben nur diese Personen das Recht, den aktuellen Aufenthaltsort anzufordern bzw. werden empfangene Ortsangaben nur von diesen Leuten angenommen (siehe *sms location request receiver* sowie *sms location respond receiver* im Abschnitt Aktivitätsdiagramme). Damit wird die Anfälligkeit für Injections verkleinert und der Benutzer kann selbst definieren, wie weit seine Privatspähre gehen soll.

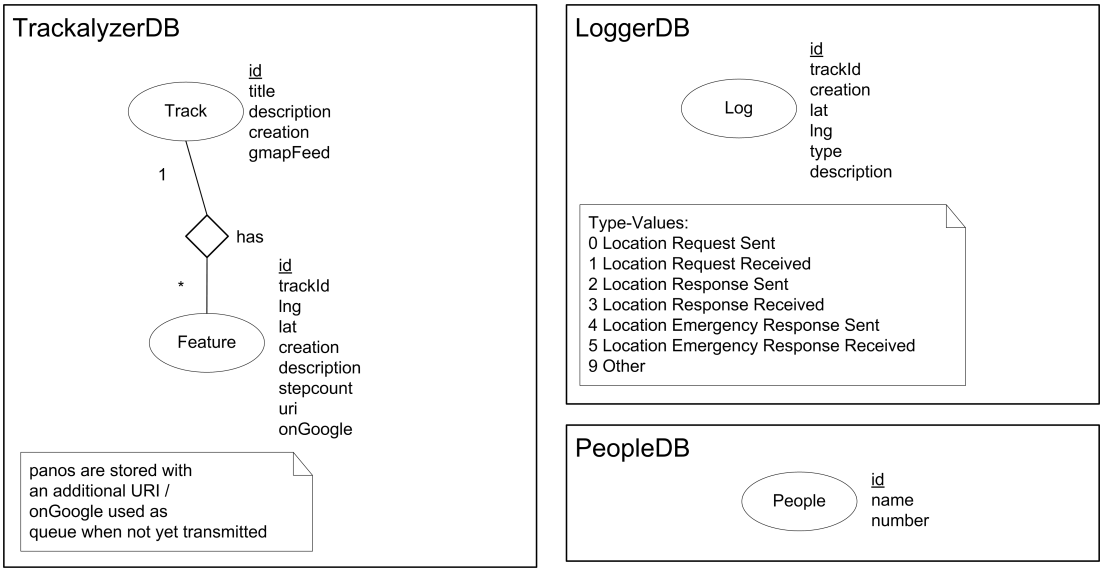

Abbildung 3.16: ER-Diagramme der Datenbanken *TrackalyzerDB*, *LoggerDB* und *PeopleDB*.

Kapitel 4

Ergebnisse

In diesem Abschnitt werden die Ergebnisse des praktischen Projekts präsentiert. Ausgehend von den Überlegungen im vorigen Abschnitt entwickelte ich einen Prototyp der geforderten Applikation, der in diesem Abschnitt vorgestellt wird. Zusätzlich zur Beschreibung des fertigen Programms zeige ich die Probleme und Einschränkungen auf, welche im Zuge der Implementierungsphase auftraten und sich teilweise gravierend auf das Gesamtergebnis auswirkten.

4.1 Programmbeschreibung

Wie bereits im Abschnitt Modellierung (Abschnitt 3.2) angedeutet, stellte es sich als sinnvoll heraus, die geforderten Funktionalitäten in zwei Packages aufzuteilen: Das Paket *Trackalyzer* sowie das Paket *Panographer* mit jeweils abgeschlossenem und sich nicht überlappendem Funktionsumfang. In meiner praktischen Ausführung ging ich noch weiter und entwickelte die Pakete jeweils als eigenständiges Programm. Sind jedoch beide Applikationen am System installiert, greifen sie nahtlos ineinander und ergänzen sich gegenseitig.

4.1.1 Trackalyzer

Das Programm wird gestartet, indem man von der Programmliste das Icon *Trackalyzer* auswählt. Daraufhin wird die grafische Bedienoberfläche angezeigt. Von dieser Oberfläche aus sind alle Funktionen des Programms erreichbar: *Start/Stop Trackalyzing*, *Visualization*, *People*, *Preferences*, *Manual Alert* und *Rescue Mode*. Zur Illustration der Startoberfläche siehe Abbildung 4.1.

Abbildung 4.1: Oberfläche des Programms *Trackalyzer*.

Start/Stop Trackalyzing

Durch Betätigen des Buttons *Start Trackalyzing* wird das kontinuierliche Aufzeichnen der zurückgelegten Wegstrecke initiiert. Je nach Voreinstellung werden dabei auch Schritte gezählt, Temperatur gemessen, Personen im Notfall automatisch alarmiert beziehungsweise die ermittelten Daten in einer Google-Maps Karte im persönlichen Google-Konto gespeichert. Der aktive Trackalyzing-Vorgang ist auch Basis für automatisch beantwortete Standortanfragen anderer Personen. Beendet wird das Trackalyzing mittels eines Klicks auf *Stop Trackalyzing*.

Visualization

Mittels Klick auf den Button *Visualization* wird die Funktionalität zur Visualisierung der gewonnenen Daten aufgerufen. Standardmäßig wird die zuletzt aufgezeichnete Wegstrecke angezeigt, mithilfe des Dropdown-Menüs im obigen Bildschirmbereich kann man auch ältere aufgezeichnete Routen auswählen und laden.

Rote Punkte kennzeichnen gemessene GPS-Koordinaten und werden sukzessive mit Linien zu einem Pfad verbunden. Durch Klicken auf einen derartigen Punkt werden je nach Voreinstellung auch zusätzliche Daten angezeigt. Dies können neben Datum und Zeit auch gezählte Schritte seit dem letzten Messpunkt, die gemessene Temperatur, oder die Höheninformation sein. Weiters werden auch die mit dem Programm *Panographer* erstellten Panoramafotos an der zugehörigen Position ein-

gezeichnet. Diese werden durch eine Kamera symbolisiert. Einen Ausschnitt einer Route mit einem Foto und zusätzlichen Informationen über einen Wegpunkt zeigt Abbildung 4.2.

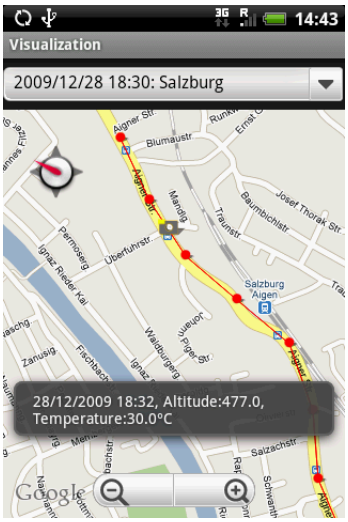

Abbildung 4.2: Beispielroute mit Foto und zusätzlichen Informationen über einen Wegpunkt.

Gingen während der Aufzeichnung der Route auch Notfallsmeldungen und Standortanfragen oder Standorte anderer Personen ein, werden auch diese in der Karte vermerkt. Dabei werden Standortdaten, die auf einer Notfallmeldung basieren mit einem Rufzeichen markiert, normale Standortdaten mit einem Blitz. Durch Klick auf ein solches Symbol werden wiederum weitere Informationen eingeblendet und die Anwendung *Radar* gestartet. Diese Anwendung erleichtert in Form eines Radars die Navigation hin zum angeklickten Ort. Sie kann vom Android Market[1] bezogen werden. Die Visualisierung derartiger Standortdaten wird in Abbildung 4.3 gezeigt.

Drückt man auf die Taste *Menu*, so kann man die aktuelle Route löschen, die Kartenansicht auf Satellitendaten umstellen sowie detaillierte Daten zur aktuellen Route aufrufen. Diese Detailansicht (siehe Abbildung 4.4) bietet genaue Informationen über Dauer, Distanz, Geschwindigkeit, Höhenmeter und andere interessante Werte.

[1] Anwendungsname *Radar* von *Mike Cleron*

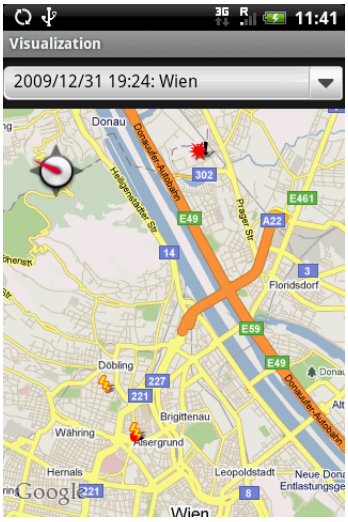

Abbildung 4.3: Eine Notfallmeldung und zwei Standorte anderer Personen.

Abbildung 4.4: Details über eine aufgezeichnete Route.

People

Unter *People* werden diejenigen Personen festgelegt, die im Notfall automatisch benachrichtigt werden sollen. Diese Personenliste kann, sofern in den Voreinstellungen so definiert, als Whitelist für Standortanfragen per SMS dienen, um nur diesen

Personen Standorte zurückzusenden wenn eine Standortanfrage eingeht. In diesem Bereich kann man Personen aus den Telefonkontakten hinzufügen und löschen. Einzelnen Personen kann man auch den eigenen Standort senden (*Send My Location*) bzw. ihren Standort anfordern (*Request Location*). Das Kontextmenü zur Auswahl dieser Funktionalitäten wird in Abbildung 4.5 gezeigt.

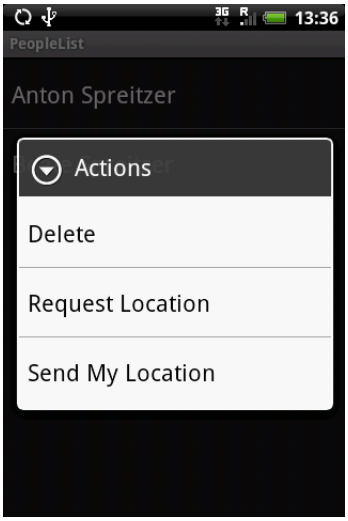

Abbildung 4.5: Kontextmenü *Löschen*, *Request Location* und *Send My Location*.

Preferences

Mittels Betätigen des Buttons *Preferences* gelangt man zur Oberfläche, um Voreinstellungen zu tätigen (siehe Abbildung 4.6). Folgende Einstellungen sind möglich:

- Die Einstellungen zu *Tracking Interval* bedeuten Zeit- bzw. Distanzintervalle, zwischen denen Wegpunkte gemessen werden.
- Die Checkbox *Allow Location Requests From Anybody?* legt fest, ob Standortanfragen von jedem oder nur von den unter People festgelegten Personen behandelt werden.
- Mit *Automatic Alertion (> 5min No Steps)?* wird definiert, ob automatisch Notfallmeldungen versendet werden, sobald mehr als fünf Minuten kein Schritt detektiert worden ist.

- Durch *Count Steps?* wird die Schrittzähler-Funktionalität aktiviert oder deaktiviert. Aufgrund technischer Einschränkungen werden Schritte nur dann gezählt, wenn das Gerät im aktiven Modus verweilt. Für nähere Informationen über dieses Problem sei auf Abschnitt 4.2 verwiesen.
- Besitzt das Endgerät auch einen Temperatursensor, kann mit dieser Einstellung auch das kontinuierliche Messen der Temperatur ein- oder ausgeschaltet werden.
- Die Checkbox *Automatic GMaps Transmission?* definiert, ob die ermittelten Daten automatisch an Google Maps übertragen werden. Auch die Anmeldedaten (Login-Name und Passwort) werden hier eingetragen.

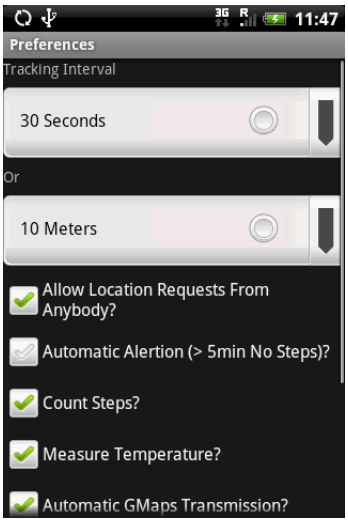

Abbildung 4.6: Oberfläche zum Definieren der Voreinstellungen.

Manual Alert und Rescue-Mode

Will man den Alarmierungsmodus manuell aktivieren, kann man dies mittels Klicks auf *Manual Alert* erreichen. Es werden dann Notfallmeldungen per SMS an alle unter People festgelegten Personen gesendet. Zusätzlich werden auch Bluetooth und WLan aktiviert und bei Auffindung eines anderen Geräts die Standortdaten übermittelt, sofern sich dieses im *Rescue-Mode* befindet. Diese Funktionalität wird auch ausgelöst, wenn die Einstellung *Automatic Alertion (> 5min No Steps)?* aktiviert ist.

Der Button *Rescue-Mode* ist das Gegenstück zu *Manual Alert*. Es werden Bluetooth und WLan aktiviert, um bei Detektion eines anderen Geräts, das sich im Alert-Modus befindet, dessen Standortdaten zu empfangen.

Aus technischen Gründen ist die Implementation von Bluetooth und WLan in diesem Prototyp nicht möglich bzw. nicht sinnvoll. Für genauere Erklärungen sei auf Abschnitt 4.2 verwiesen.

SMS Protokoll des Standort-Anfrage-Systems

Das Standort-Anfrage-System verwendet SMS (siehe Abschnitt 2.4) als Kommunikationstechnologie. Die Nachrichten, welche zur Standort-Anfrage und als Standort-Antwort gesendet werden, sind dabei nicht nur maschinen- sonder auch menschenlesbar. Damit können auch Personen den Standort anfragen, welche das Trackalyzer-Programm nicht benutzen. Die Nachricht, um den Standort von einer Person anzufragen, lautet `where?`. Sofern der Empfänger gerade seine Route aufzeichnet und den Sender unter People eingetragen oder die Einstellung *Allow Location Requests From Anybody?* gesetzt hat, wird dem Anfragenden der Standort des Empfängers gesendet. Diese Nachricht hat das Format `iam@(xx.xxx/xx.xxx)` und stellt die Position in Breiten- und Längengraden dar. Ist der Nachricht zusätzlich die Zeichenkette `!emergency!` angehängt, zeigt dies an, dass der Empfänger in einer Notsituation ist.

Erfüllung der Anforderungen

Damit sind in diesem Programm die Anforderungen (siehe Kapitel 3) bezüglich Standort-Tracking, Standort-Anfrage-System und Visualisierung erfüllt. Die Anforderung zur Erstellung eines Schrittzählers kann als teilweise erfüllt angesehen werden. Technische Einschränkungen (siehe Abschnitt 4.2) bewirken, dass der Akzelerometersensor im Standby-Modus des Geräts abgeschaltet wird. Eine sinnvolle Anwendung des Schrittzählers ist damit nicht möglich. In Verbindung mit diesem Sachverhalt steht auch die Anforderung zur automatischen Alarmierung, wenn eine gewisse Zeit kein Schritt detektiert wurde. Aus oben genanntem Grund ist auch diese Anforderung nur teilweise, also nur wenn das Gerät aktiv ist, erfüllbar. Des weiteren ist es aus technischen Begrenzungen nicht möglich, die Kommunikationskanäle Bluetooth und WLan zur Alarmierung zu nutzen. Die Anforderung betreffend Visualisierung der gesammelten Daten am Gerät und im Internet ist erfüllt.

4.1.2 Panographer

Indem man in der Programmliste das Programm Panographer auswählt, wird die Applikation zum Erstellen von Panoramafotos gestartet und es erscheint die graphische Benutzeroberfläche (siehe Abbildung 4.7): Im linken Bildschirmbereich ist die Funktionsleiste angebracht, rechts die Live-Vorschau der Kamera.

Will man ein Panorama erstellen, genügt das Betätigen der Taste *Record Pano*, wodurch das erste Foto der Szene aufgenommen wird. Schwenkt man die Kamera nun langsam von links nach rechts über die Landschaft, so werden automatisch weitere Fotos erstellt. Dieser Vorgang wird durch einen Klick auf den Knopf *Stop* abgeschlossen. Ist man mit dem aufgenommenen Blickwinkel nicht zufrieden, kann durch Drücken von *Record Pano* der Aufnahmevorgang erneut initiiert werden. Abschließend wird durch Drücken des Knopfs *Render* das Panoramafoto erstellt, auf der Speicherkarte gespeichert und in der aktuellen Route des Programms Trackalyzer vermerkt, sofern es gerade eine Route aufzeichnet.

Zusätzlich ist es auch möglich, ein einzelnes Foto zu erstellen. Dazu betätigt man die Taste *Take Photo*. Auch dieses wird auf der Speicherkarte gespeichert und an das Programm Trackalyzer weitergeleitet. Will man lediglich

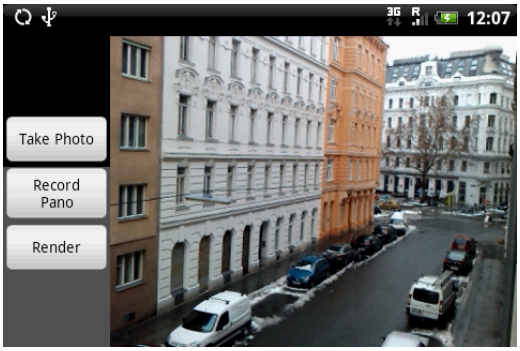

Abbildung 4.7: Oberfläche der Applikation *Panographer*.

Erfüllung der Anforderung

Die Anforderung, Panoramafotos durch Schwenken des Geräts über die Landschaft zu erstellen, kann aus technischer Sicht als erfüllt betrachtet werden, wenn auch mit der Einschränkung, dass man die Kamera sehr langsam schwenken muss und das Aufnehmen der Fotos sehr träge funktioniert. Die Gründe werden in Abschnitt 4.2

ausgeführt. Die aufgenommenen Einzelfotos können anschließend durch Betätigen des Knopfs *Render* zu einem Gesamtfoto verarbeitet (für Testergebnisse siehe Abschnitt 4.3) und im Trackalyzer eingebunden werden.

4.1.3 Visualisierung unter Google Maps

Ist im *Trackalyzer* die Option *Automatic GMaps Transmission?* aktiviert und sind gültige Benutzerdaten eines Google Accounts eingetragen, so werden die ermittelten Daten in Echtzeit mithilfe einer mobilen Datenanbindung an das persönliche Google Maps Konto gesendet. Für jeden Trackalyzing-Vorgang wird eine Karte angelegt und die ermittelten Wegpunkte mit Datums- und Zeitangabe inklusive der zusätzlichen Informationen (z.B. gezählte Schritte, Temperaturdaten, ...) in diese übertragen. Werden während der Routenaufzeichnung mit dem Programm *Panographer* Panoramafotos erstellt, so werden auch diese in der Karte eingezeichnet. Dazu werden die Fotos zuerst an die Ablagebox von *PicasaWeb Albums* gesendet und daraufhin an der georeferenzierten Stelle in der aktuellen Landkarte eingebunden. Abbildung 4.8 zeigt die Route aus Abbildung 4.2 mit einigen ermittelten Wegpunkten sowie ein georeferenziertes Panorama in Google Maps.

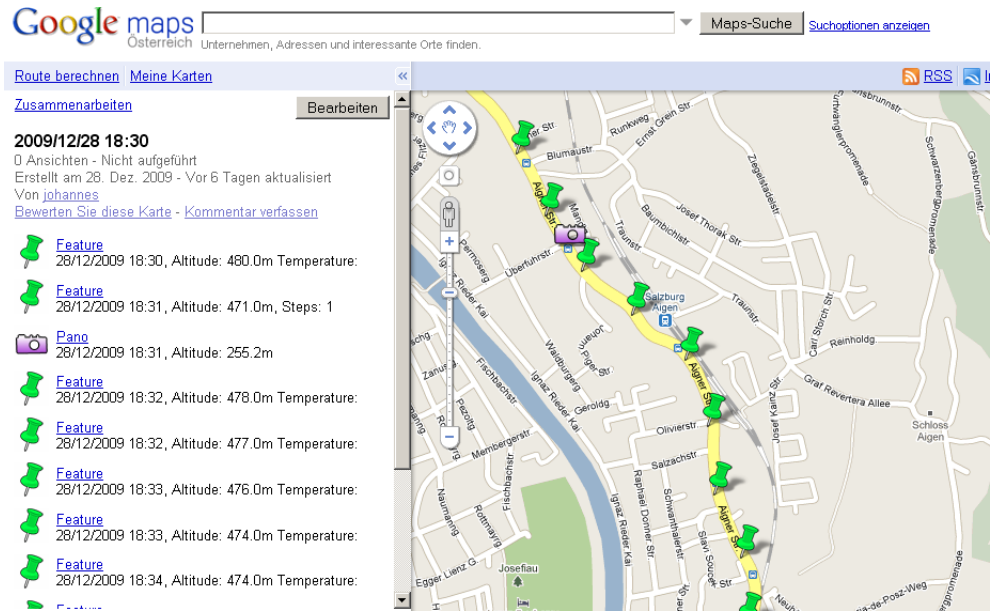

Abbildung 4.8: Repräsentation einer Route in Google Maps.

4.2 Einschränkungen

Diese Sektion behandelt Probleme und Einschränkungen von Android, welche im Zuge der praktischen Umsetzung meines Projekts aufgetreten sind.

4.2.1 Akzelerometer

Eine große Einschränkung betrifft die Implementierung des Schrittzählers (siehe Abschnitt 2.3). Das Android-System teilt die Sensordaten des Akzelerometers mithilfe eines `SensorEvent` mit, welche von der eigenen Applikation mittels eines `SensorEventListener` abgefangen und in der Methode `onSensorChanged()` weiterverarbeitet werden. Auf den mir zur Verfügung stehenden Testgeräten wird diese Methode allerdings nicht mehr aufgerufen, sobald das Gerät in den Standby-Modus wechselt. Meine Umgehungslösung ist, das Gerät mit einem *wake-lock* anzuweisen, nicht automatisch in den Standby-Modus zu wechseln. Dann jedoch bleibt das Display erleuchtet und reagiert auf etwaige Touch-Eingaben. Gerade für die Funktionalität eines Schrittzählers, der meist dann aktiv ist, wenn das Gerät selbst nicht verwendet wird und sich daher in einer Tasche befindet, ist dieser Ansatz sehr unbefriedigend. Noch dazu wird für die Erleuchtung des Bildschirms unnötig viel Energie verbraucht, was die maximale Laufzeit einer Routenaufzeichnung stark reduziert.

Dieses Problem wird auch auf der offiziellen Issue-Liste von Android angeführt und scheint hauptsächlich Geräte mit Android Version 1.5 oder neuer zu betreffen.[1]

4.2.2 Alarmierung und Rettung über PANs

Wie bereits in Abschnitt 2.1.6 erwähnt, gibt es in der im praktischen Projekt verwendeten Android Version 1.5 noch keinen programmierbaren Bluetooth-Stack. Damit kann dieser Kommunikationskanal für die Alarmierungs- und Rettungsfunktionalität des Trackalyzers nicht verwendet werden. Daher war mein Lösungsansatz, anstatt von Bluetooth WLan zu verwenden, um diese Funktionalität trotzdem zu gewährleisten. Der `WifiManager` lässt es jedoch nicht zu, eine WLan-Verbindung von einem Endgerät zu einem anderen Endgerät ohne Infrastruktur (ein so genanntes *Ad-Hoc Netzwerk*) zu erstellen. Die Alarmierungs- und Rettungsfunktionalität über PANs kann damit aus technischen Gründen nicht umgesetzt werden.

[1] Vgl. [40]

Auch über das *Ad-Hoc-Networking*-Problem existiert ein Eintrag in der offiziellen Issue-Liste Androids.[1]

4.2.3 Medienverarbeitung

Eine große Einschränkung betrifft die Möglichkeit zur Medienverarbeitung in Android. Die Anforderung zur Erstellung der Panoramafotos war, die Ausgangsdaten durch einfaches Schwenken der Kamera über die Landschaft zu gewinnen. Mein ursprünglicher Ansatz war dabei, ein Video aufzunehmen und daraus die infrage kommenden Frames zu extrahieren und dem Stitching-Prozess zuzuführen. Obwohl es ein Leichtes ist, Videosequenzen aufzunehmen, bieten die Media-APIs keine Möglichkeit, einzelne Frames aus einem Video zu entnehmen. Am Android-Gerät ist daher der videobasierte Ansatz zur Erstellung von Panoramafotos unmöglich. Abhilfe würde lediglich schaffen, ein aufgezeichnetes Video eines Panoramas an einen externen Server zu senden, der die Verarbeitung erledigt und das fertig gerenderte Foto an das Gerät zurücksendet.

Da der videobasierte Ansatz keine Lösung versprach, versuchte ich, die Ausgangsdaten mithilfe von Fotos zu akquirieren. Meine Idee war, die Sensordaten des digitalen Kompass dazu zu verwenden, automatisch beispielsweise alle 20° Grad ein Foto zu schießen und damit die Anforderung *Schwenken der Kamera über die Landschaft* zu erfüllen. Doch dabei tritt das Problem auf, dass der Aufnahmevorgang eines Fotos asynchron vonstatten geht.[2] Damit ist nicht garantiert, wann genau das Foto aufgenommen wird, zumal sich die Auslöseverzögerung auch abhängig von der Belichtung und der gewünschten Größe des Fotos ändert. Indem man bei der Aufnahme nur sehr langsam über das Panorama schwenkt kann man jedoch auch diese Einschränkung einigermaßen umgehen. Des weiteren trat das Problem auf, dass nach der Aufnahme des ersten Einzelfotos der Vorschaubildschirm nur noch mit sehr geringer Framerate bespielt wurde und nach ein paar Sekunden zu flimmern begann. Die Lösung dieses Problems lag in der automatischen Neuinitialisation der Kamera nach jedem aufgenommenen Einzelfoto. Die anschließende Verarbeitung der Bilddaten ist nur noch durch die Hardwareausstattung des spezifischen Endgeräts beschränkt. Bei einer Einzelbildgröße von 320x240 Pixel ist die Verarbeitung zu einem Panoramafoto ohne Ressourcenprobleme möglich.

[1] Vgl. [41]
[2] Vgl. [42]

4.3 Testergebnisse

4.3.1 Test-Hardware

Zur Entwicklung und zum Testen der beiden Programme Trackalyzer und Panographer standen mir zwei Android-Mobiltelefone zur Verfügung: *HTC Hero* und *HTC Tattoo*.

- Das Gerät HTC Hero hat einen Qualcomm® MSM7200A™ Prozessor mit 528 MHz sowie 512 MB ROM und 288 MB RAM. Die Batteriekapazität beträgt 1350 mAh. Es läuft unter Android-Version 1.5.[1]
- Das Gerät HTC Tattoo wird von einer Qualcomm® MSM7225™ CPU mit 528 MHz angetrieben und verfügt über 512MB ROM bzw. 256MB RAM. Seine Batteriekapazität beträgt 1100 mAh. Es verwendet Android-Version 1.6.[2]

Durch die ältere Android-Version 1.5 wurde gleichzeitig die minimale SDK-Version vorgegeben, für die ich die Applikationen erstellen musste, damit sie auf beiden Geräten ausführbar sind.

4.3.2 Trackalyzer

Um einen Eindruck über die maximale Batterielaufzeit bei Einsatz der Applikation Trackalyzer zu erhalten, habe ich sie in einem Echtszenario in Wien getestet. Mit vollem Akku und den Einstellungen Tracking alle 100 Meter bzw. 2 Minuten, deaktivierter Schrittzähler, deaktivertes Thermometer und aktivierte Übertragung an Google Maps startete ich den Trackalyzing-Vorgang. Ich war vier Stunden und vierzig Minuten unterwegs, machte auch ein Foto mit dem Programm Panographer und als ich die Routenaufzeichnung beendete, hatte der Akku noch 46% verbleibende Kapazität. Ich gehe daher davon aus, dass mit einer Akkufüllung ein Programmeinsatz von bis zu sieben Stunden möglich ist. Dieser Wert ist jedoch von den Einstellungen des Trackalyzers, von zusätzlichen im Hintergrund laufenden Services, vom Gerät, von der Akkukapazität und von der Qualität der Netzversorgung abhängig. Er kann daher nur als ungefährer Richtwert gelten. Die zurückgelegte Route mit den Repräsentationen am Gerät und in Google Maps sind in Abbildung 4.9 und Abbildung 4.10 ersichtlich.

[1] Vgl. [43]
[2] Vgl. [44]

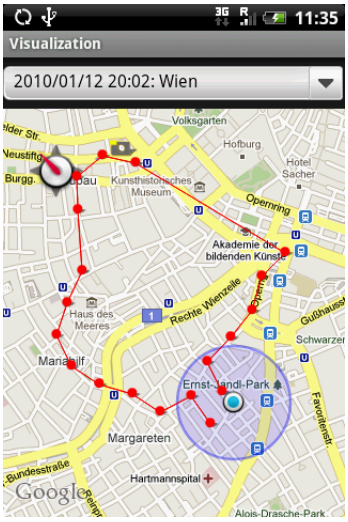

Abbildung 4.9: Repräsentation der Testroute am Gerät.

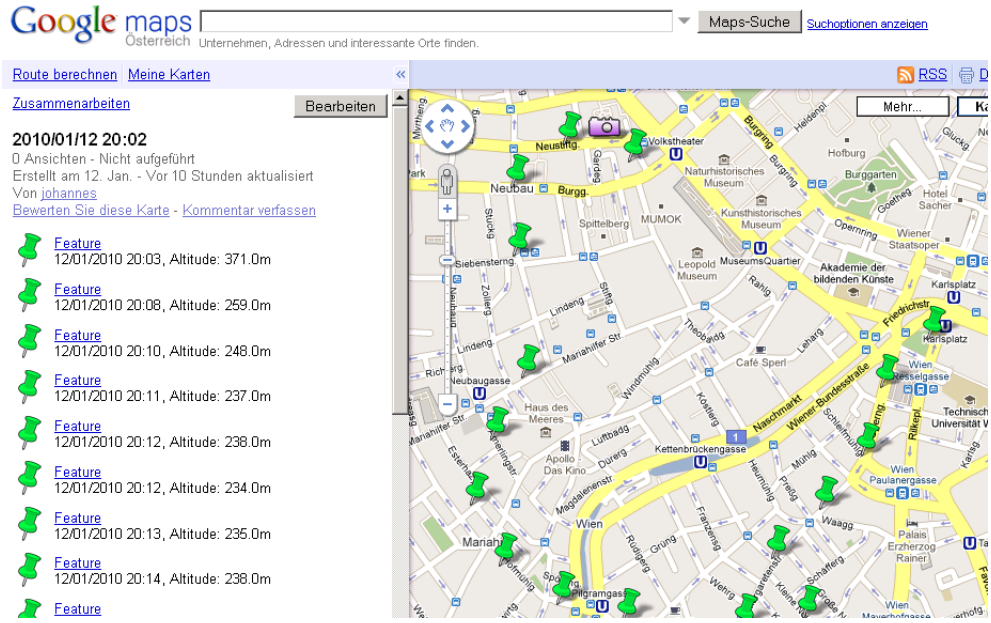

Abbildung 4.10: Repräsentation der Testroute in Google Maps.

4.3.3 Panographer

Zur Abschätzung des Laufzeitverhaltens habe ich zwei Panoramafotos auf beiden mir zugänglichen Android-Geräten rendern lassen. Das erste besteht aus sechs Einzelbilder mit einer jeweiligen Größe von 320x240 Pixel. Das Ergebnisbild ist 661x349 Pixel groß und in Abbildung 4.12 zu sehen. Das zweite besteht aus drei Einzelbildern mit derselben Größe und wird zu einem Bild mit der Größe von 553x257 Pixel zusammengefügt. Es ist in Abbildung 4.11 sichtbar. Anhand der Ergebnisfotos sieht man, dass die sich überlappenden Regionen ungenau gefunden werden. Der Grund hierfür liegt im Verfahren nach diesen zu suchen.

In Tabelle Tabelle 4.1 ist die jeweils benötigte Rechenzeit angeführt. Man erkennt, dass das HTC Hero diese Aufgabe etwas schneller erledigt als das HTC Tattoo. Dies kann an der unterschiedlichen Betriebssystem-Version oder am etwas größeren Arbeitsspeicher des HTC Hero liegen. Die Verarbeitungszeit steigt direkt proportional mit der Anzahl der Einzelfotos. Auch wenn die Ausführungszeiten der zwei Geräte untereinander etwas auseinander liegen, kann man den ungefähren Rechenzeitaufwand grob mit ca. viereinhalb Sekunden pro Einzelfoto abschätzen.

Gerät	3 Einzelfotos	ø pro Foto	6 Einzelfotos	ø pro Foto
HTC Hero	10 Sek	3.3 Sek	23 Sek	3.8 Sek
HTC Tattoo	17 Sek	5.6 Sek	28 Sek	4.6

Tabelle 4.1: Benötigte Rechenzeiten für die Panoramaerstellung.

Abbildung 4.11: Aus drei Einzelfotos aufgebautes Panoramafoto.

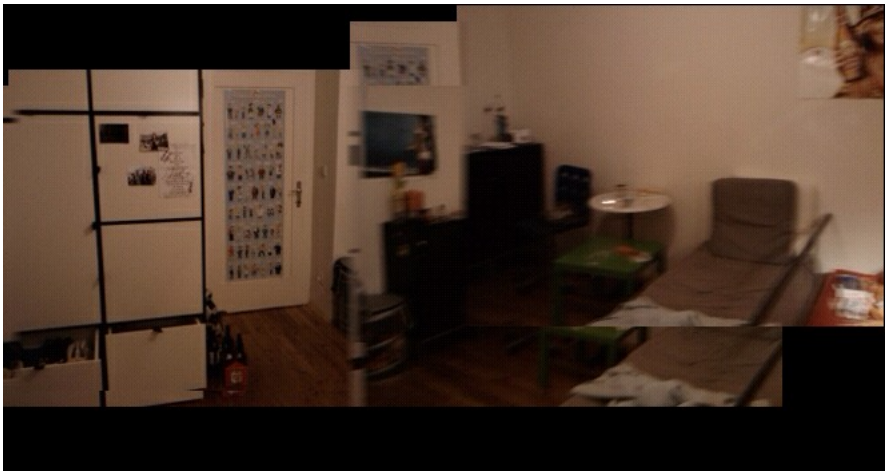

Abbildung 4.12: Aus sechs Einzelfotos aufgebautes Panoramafoto.

Kapitel 5

Zusammenfassung

Diese Diplomarbeit behandelte die Software-Entwicklung und Medienverarbeitung anhand eines konkreten praktischen Projekts unter dem Betriebssystem Android. Dazu wurden zuerst die zentralen Konzepte von Android erarbeitet und danach die Hintergründe der im Projekt verwendeten Technologien näher beleuchtet. Anschließend wurde das Projekt mittels gängiger Verfahren einer Anforderungsanalyse unterzogen, die Software-Teile modelliert und damit die Programmarchitektur festgelegt. Auf dieser Basis erfolgte die Implementierung des Programms mit dem Resultat eines lauffähigen Prototyps. Abschließend wurden die Ergebnisse präsentiert.

Die Ergebnisse zeigten, dass ein Großteil der Funktionalitäten der ursprünglichen Projektidee unter Android in Version 1.5 umgesetzt werden konnten. Nicht möglich ist das Nutzen von Bluetooth, sowie von Ad-Hoc-WLan-Verbindungen. Ebeno unmöglich ist das Nutzen des Akzelerometersensors wenn sich das Gerät im Standby-Modus befindet. Der implementierte Schrittzähler ist damit technisch durchführbar und funktionsfähig, praktisch jedoch nur einsetzbar, wenn man das Gerät veranlasst, aktiv zu bleiben. Sein Einsatz ist daher für mobile Szenarien nicht sinnvoll, wo Energie nur in sehr begrenztem Ausmaß vorhanden ist. Dies betrifft in der Folge auch die Funktionalität, automatisch Notfallmeldungen abzusetzen, wenn eine gewisse Zeit kein Schritt detektiert wurde. Diese Restriktionen könnten sich jedoch durch zukünftige Entwicklungen bezüglich Akkutechnologie und Stromverbrauch relativieren. Bezüglich Medien- und Bildverarbeitung zeigte sich, dass unter Android nur sehr spärliche Möglichkeiten bereitstehen. So ist es beispielsweise nicht möglich, Standbilder aus Videos zu extrahieren. Bilddaten können daher nur aus Fotos einer weiteren Verarbeitung zugeführt werden. Im Zuge der Panoramafunktionalität musste ich somit auf Fotos als Ausgangsmaterial zurückgreifen. Das

Resultat der Panoramaerstellung zeigte die grundsätzliche Eignung des Systems, Bilddaten zu verarbeiten. Wegen der - verglichen mit herkömmlichen Computern - begrenzten Hardwareressourcen konnte ich jedoch nur sehr einfache Features verwenden, um überlappende Regionen zwischen zwei Einzelfotos zu finden, was sich negativ auf das visuelle Ergebnis des Stitching-Vorgangs auswirkte. Zur visuellen Darstellung der Daten standen sowohl unter Android als auch im Internet ausgereifte Methoden zur Verfügung, auf die ich zurückgriff. Unter Android bot das Oberflächenelement MapView eine gut erweiterbare Möglichkeit, eine Landkarte mit den eigenen Daten zu versehen. Durch die Übertragung der Daten an Google Maps im Internet ergab sich zudem auch die Möglichkeit zur Visualisierung in einem Webbrowser sowie zur Datenspeicherung außerhalb des Geräts.

Zur Bildverarbeitung ist anzumerken, dass es für Android (noch?) keine Bibliotheken gibt, die spezielle Aufgaben der Bildverarbeitung übernehmen. So wären Funktionalitäten wünschenswert, die verschiedenartige Features aus Bilddaten extrahieren, Histogramme bilden können, u.s.w. Solche Bibliotheken wären hilfreich, um die Resultate des Panographers zu verbessern. Ein anderer Ansatz wäre, die Android-Geräte als *Thin-Clients* anzusehen. Man würde die Geräte dann lediglich zur Datenein- und -ausgabe einsetzen und etwaige aufwendige Rechenoperationen in externe Server auslagern. Dies wäre aufgrund der guten Netzwerkanbindung ohne gravierende Probleme möglich und man könnte die angeführten Restriktionen bezüglich Medienverarbeitung in Android umgehen.

Das Betriebssystem Android habe ich als sehr durchdacht und stabil empfunden. Das zugehörige SDK ist sehr gut dokumentiert und die Programmierlernkurve ist - Javakenntnisse vorausgesetzt - sehr steil. Es bietet ein modernes System, das die derzeit in einem Mobiltelefon integrierten Technologien konsistent in einer API-Sammlung vereint. Sehr gefallen hat mir die Offenheit von Android. Diese wird hauptsächlich durch das Konzept der Intents gewährleistet. Mittels Intents ist es einerseits möglich, ein Programm aus einzelnen voneinander entkoppelten Komponenten zu erstellen und andererseits Drittentwicklern die Chance zu geben, eigene Komponenten mitzunutzen oder auf eingetretene Events zu reagieren. Zusätzlich wird die Offenheit auch durch den Aspekt unterstrichen, dass die Quellcodes von Android unter einer Open-Source-Lizenz veröffentlicht werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Literaturverzeichnis

Bücher

[1] Arno Becker, Marcus Pant, *Android, Grundlagen und Programmierung*, dpunkt.verlag, 2009.

[2] Ed Burnette, *Hello, Android - Introducing Google's Mobile Development Platform*, Pragmatic Bookshelf, 2008.

[3] Reto Meier, *Professional Android Application Development*, Wiley Publishing. Inc., 2009.

[4] Heiko Mosemann, Kose Matthias, *Android - Anwendungen für das Handybetriebssystem erfolgreich programmieren*, Carl Hanser Verlag, 2009.

[5] Rick Rogers, John Lombardo, Zugurd Mednieks, Blake Meike, *Android Application Development*, O'Reilly Media, Inc., 2009.

Artikel

[6] Martin Mladenov, Michael Mock, In: *A Step Counter Service for Java-Enabled Devices Using a Built-In Accelerometer*, ACM International Conference Proceeding Series; Vol. 385, Pages 1-5, 2009.

[7] I. K. Adusei, K. Kyamakya, F. Erbas, In: *Location-Based Services: Advances and Challenges*, Canadian Conference on Electrical and Computer Engineering 2004 Vol. 1, Pages 1-7, 2.-5. Mai 2004.

[8] Peter Fröhlich, Rainer Simon, Lynne Baillie, Joi Roberts, Roderick Murray-Smith, , In: *Mobile Spatial Interaction*, CHI '07 extended ab-

stracts on Human factors in computing systems, Pages 2841 - 2844, 28.April-3.Mai 2007.

[9] Jiang Ling, Li Bin, Gong Jianya, Min Min, In: *GeoReferencing the Semantic Web Based on Geoontology*, IEEE International Conference on Geoscience and Remote Sensing Symposium 2006, Pages 1545 - 1548, 31. Juli - 4. August 2006.

[10] Victor C.M. Leung, Terrence Wong, Peyman TalebiFard, In: *Breaking the Silos - Access and Service Convergence over the Mobile Internet*, Proceedings of the 11th international symposium on Modeling, analysis and simulation of wireless and mobile systems 2008, Pages 286-293, 27.-31.Oktober 2008.

[11] Earl Oliver, In: *Exploiting the Short Message Service as a Control Channel in Challenged Network Environments*, CHANTS'08, Pages 57-64, 15. September 2008.

[12] Myra Dideles, In: *Bluetooth: a technical overview*, Crossroads Volume 9, Issue 4, Juni 2003.

[13] Ciou-Ting Hsu, Tz-Hung Cheng, Rob A. Beuker, Jyh-Kuen Horng, In: *Feature-Based Video Mosai*, International Conference on Image Processing, 2000. Proceedings. Volume 2, Pages 887 - 890, 10-13 September 2000.

[14] Richard Szeliski, Heung-Yeung Shum, In: *Creating Full View Panoramic Image Mosaics and Enviroment Maps*, International Conference on Computer Graphics and Interactive Techniques, Proceedings of the 24th annual conference on Computer graphics and interactive techniques, Pages 251 - 258, August 1997.

Internet

[15] Quelle Abbildung 2.1, letzter Besuch am 12.10.2009
`www.openhandsetalliance.com/media_room.html`

[16] Quelle Abbildung 2.2, letzter Besuch am 12.10.2009
`www.android.com/goodies/`

[17] Android is now available as open source, letzter Besuch am 14.10.2009
`source.android.com/posts/opensource`

[18] What is the Android NDK?, letzter Besuch am 16.10.2009
`developer.android.com/sdk/ndk/1.6_r1/index.html`

[19] Hardware products running Android, letzter Besuch am 14.10.2009
`en.wikipedia.org/w/index.php?title=Android_(operating_system)&oldid=319668017`

[20] A Note on Google Apps for Android, letzter Besuch am 14.10.2009
`android-developers.blogspot.com/2009/09/note-on-google-apps-for-android.html`

[21] Designing for Responsiveness, letzter Besuch am 15.10.2009
`developer.android.com/guide/practices/design/responsiveness.html`

[22] OpenIntents, letzter Besuch am 16.10.2009
`www.openintents.org`

[23] Android API Reference, letzter Besuch am 20.10.2009
`developer.android.com/reference/packages.html`

[24] Some information on APIs removed in the Android 0.9 SDK beta , letzter Besuch am 20.10.2009
`android-developers.blogspot.com/2008/08/some-information-on-apis-removed-in.html`

[25] Android Market-Hilfe, letzter Besuch am 20.10.2009
`market.android.com/support/bin/topic.py?topic=15866`

[26] Announcing Android 2.0 support in the SDK!, letzter Besuch am 28.10.2009
`android-developers.blogspot.com/2009/10/announcing-android-20-support-in-sdk.html`

[27] iPhone OS, letzter Besuch am 31.10.2009
`en.wikipedia.org/w/index.php?title=IPhone_OS&oldid=322734125`

[28] Java Platform, Micro Edition, letzter Besuch am 31.10.2009
`de.wikipedia.org/w/index.php?title=Java_Platform,_Micro_Edition&oldid=65566408`

[29] Microsoft Windows Mobile, letzter Besuch am 31.10.2009
de.wikipedia.org/w/index.php?title=Microsoft_Windows_Mobile&oldid=66227549

[30] Palm webOS, letzter Besuch am 31.10.2009
de.wikipedia.org/w/index.php?title=Palm_webOS&oldid=65792577

[31] BlackBerry OS, letzter Besuch am 31.10.2009
en.wikipedia.org/w/index.php?title=BlackBerry_OS&oldid=320448135

[32] Symbian OS, letzter Besuch am 31.10.2009
de.wikipedia.org/w/index.php?title=Symbian_OS&oldid=65887679

[33] What are the Kits?, letzter Besuch am 31.10.2009
developer.symbian.org/wiki/index.php/What_are_the_Kits%3F

[34] Georeferenzierung, letzter Besuch am 4.11.2009
http://de.wikipedia.org/w/index.php?title=Georeferenzierung&oldid=64605509

[35] David Dewey, Virtual Port Townsend Technical Details, letzter Besuch am 7.1.2010
http://www.ddewey.net/pics/vpt/technical/

[36] Package com.google.android.maps, letzter Besuch am 11.11.2009
code.google.com/intl/de-DE/android/add-ons/google-apis/reference/index.html

[37] What is the Google Data Protocol?, letzter Besuch am 11.11.2009
code.google.com/intl/de-DE/apis/gdata/

[38] What is the Maps Data API?, letzter Besuch am 11.11.2009
http://code.google.com/intl/de-DE/apis/maps/documentation/mapsdata/

[39] What is the Picasa Web Albums Data API?, letzter Besuch am 11.11.2009
http://code.google.com/intl/de-DE/apis/picasaweb

[40] Issue 3708: OnSensorChanged() is no longer called in standby mode since last Firmware upgrade., letzter Besuch am 4.1.2010
http://code.google.com/p/android/issues/detail?id=3708

[41] Issue 82: wifi - support ad hoc networking, letzter Besuch am 4.1.2010
`http://code.google.com/p/android/issues/detail?id=82`

[42] Android API Reference - Camera, letzter Besuch am 4.1.2010
`http://developer.android.com/reference/android/hardware/Camera.html`

[43] HTC - Products - HTC Hero - Specification, letzter Besuch am 11.01.2010
`http://www.htc.com/europe/product/hero/specification.html`

[44] HTC - Products - HTC Tattoo - Technische Daten, letzter Besuch am 11.01.2010
`http://www.htc.com/de/product/tattoo/specification.html`