Wissenschaftliche Arbeit zur Erlangung des
akademischen Grades Bachelor of Science

Evaluierung der Android-Plattform anhand einer Referenzanwendung

vorgelegt von
Sebastian Kroop

Betreuer:

Prof. Dr.-Ing. Thomas Preuß
Dipl.-Inf. (FH) Jonas Brüstel

Fachhochschule Brandenburg
Fachbereich Informatik und Medien

Brandenburg, den 2. Februar 2009

Inhaltsverzeichnis

1 Einleitung

Die Android–Plattform (Android) der Firma Google ist eine quelloffene (Open Source) Softwareplattform für mobile Endgeräte. Sie enthält ein Betriebssystem, Middleware und Standardanwendungen und ist mit einer erweiterten und optimierten virtuellen Maschine für Java–Anwendungen unterlegt.

1.1 Historie

Die Ambitionen von Google in den Markt der Betriebssysteme für Smartphones einzutreten, waren lange Zeit Gegenstand von Spekulationen (vgl. [McK06]).

Im Juli 2005 kaufte Google das junge Unternehmen Android Inc. aus Palo Alto/Kalifornien (vgl. [Elg05]), welches Software für Mobiltelefone entwickelte. Das Wall Street Journal berichtete zu diesem Zeitpunkt, dass von Google die Entwicklung einer Anwendung der Google–Suche für mobile Endgeräte beabsichtigt wird. Hierbei will Google den Nutzern einen Zugang zu spezifischen Inhalten gebührenpflichtiger Dienste für Mobiltelefone anbieten (vgl. [Sha07]). Weiterhin berichtete die Information Week im September 2007 aufgrund einer Studie des Marktforschungsinstitutes Evalueserve, dass Google mehrere Patente im Bereich der Mobiltelefonie angemeldet hat (vgl. [Cox07], [Cla07]).

Durch die Gründung der Open Handset Alliance (OHA) am 5. November 2007 wurden die Spekulationen dann beendet (vgl. [Ope07]). Die OHA umfasst 34 Mitglieder, die von Google geführt werden. In dieser Gruppe sind Mobiltelefonentwickler, Anwendungsentwickler, Funknetzbetreiber und Chipentwickler vertreten. Am Tag der Gründung der OHA erfolgte zugleich auch die Veröffentlichung von Android (vgl. [Ope07]). Das erste Softwareentwicklungssystem, das sogenannte Software Development Kit (SDK), wurde am 12. November 2007 für Entwickler bereitgestellt (vgl. [Ihl07]). Das erste kommerziell verfügbare Telefon auf Basis von Android war das T–Mobile G1, das auch unter der Bezeichnung HTC Dream bekannt ist. Die Federal Communications Commission (FCC) zertifizierte die-

ses Gerät in den USA am 18. August 2008 (vgl. [Ric08]) und ab Oktober 2008 war es dann am Markt verfügbar (vgl. [Bro08]).

Auf dem Markt vergleichbarer Systeme für Smartphones haben sich parallel weitere Produkte wie z. B. Symbian OS, Windows Mobile, iPhone OS und OpenMoko etabliert. Damit war und ist Google einer starken Konkurrenz ausgesetzt. Für Android waren deshalb von Beginn an Alleinstellungsmerkmale erforderlich, um sich erfolgreich in diesem Markt etablieren zu können. Android wurde auf der Grundlage des plattformunabhängigen Betriebssystems Linux konzipiert. Android beinhaltet eine umfangreiche Klassenbibliothek sowie die virtuelle Maschine Dalvik. Weitere Schichten oberhalb davon sind ein speziell für Android entwickeltes Anwendungsframework und die Anwendungsschicht. Die Quellen von Android sind offengelegt. Bereits am 21. Oktober 2008 veröffentlichte Google den gesamten Quellcode unter der Apache–License (vgl. [Bor08]). Damit ist ein umfassender Einblick in die Funktionsweise gegeben. Bei Symbian OS, Windows Mobile, iPhoneOS ist eine solche Transparenz nicht gegeben. Die Einbindung offener Standards und die Veröffentlichung eines Software Development Kit (SDK) ermöglichen darüber hinaus eine komfortable Entwicklung von Anwendungen für Android.

Das SDK verfügt über eine umfangreiche Auswahl an Entwicklungswerkzeugen (vgl. [Goo08a]). Im Lieferumfang sind ein Fehleranalyseprogramm, Bibliotheken, ein auf QEMU beruhender Telefon–Emulator, Dokumentationen, Beispielcodes und Übungsbeispiele enthalten. Zurzeit wird die Entwicklungsplattform von verschiedenen Betriebssystemen auf x86-Computern unterstützt.

Voraussetzung für die Entwicklung von Android–Anwendungen ist das Vorhandensein folgender Komponenten:

- Java Development Kit ab Version 5
- Apache Ant ab Version 1.6.5
- optional Python ab Version 2.2

Die offiziell unterstützte Entwicklungsumgebung (IDE) ist Eclipse ab Version 3.2 in Verbindung mit dem Android Development Tool (ADT) als Plugin. Ferner ist es möglich, mit den mitgelieferten Kommandozeilenwerkzeugen Anwendungen für Android zu erstellen und zu debuggen. Eine erste Vorschau auf SDK wurde am 12. November 2007 veröffentlicht. Am 18. August 2008 folgte dann mit Android SDK 0.9 die erste Betaversion. Diese Version des SDK stellte eine Aktualisierung und Erweiterung der API, verbesser-

te Entwicklungswerkzeuge und ein aktualisiertes Design für den Grundbildschirm zur Verfügung. Die erste gültige Version 1.0 des SDK wurde schließlich am 23. September 2008 veröffentlicht. Hier wurden hauptsächlich Fehler der Betaversion beseitigt und einige neue Features hinzugefügt. Sie beinhaltete zusätzlich wiederum einige Änderungen der API.

1.2 Zielsetzung

Mit dieser Bachelorarbeit sollen die Möglichkeiten von Android aufgezeigt werden und anschließend anhand der prototypischen Umsetzung einer Referenzanwendung evaluiert werden. Durch die Referenzanwendung sollen Wegstrecken aufgezeichnet werden. Die Referenzanwendung bezieht sich auf ein Teilproblem des Projektes „Waldnavigation“ der Fachhochschule Brandenburg. Mit der Evaluierung ist zu prüfen, ob die Plattform für eine Verwendung im Rahmen des Projektes geeignet ist.

1.3 Motivation

Ziel des Projektes „Waldnavigation“ ist es, durch ein spezielles Navigationssystem auf mobilen Endgeräten die Effizienz von Holztransporten im Wald zu steigern.

Aus den bisherigen Erkenntnissen zu Android werden folgende Vorteile erwartet:

Android verwendet Linux und Java. Damit ist eine Portabilität auf Geräte unterschiedlicher Hersteller denkbar. Java ist eine aufgrund seiner Vorzüge weit verbreitete Programmiersprache. Android kann einfach und intuitiv bedient werden.

2 Android

2.1 Architektur

Android ist mit einem Linux–Kernel als Betriebssystem und einer erweiterten und optimierten virtuellen Maschine für Java–Anwendungen unterlegt. Die Vereinigung der Vorteile beider Technologien ist für die Leistungsfähigkeit und damit für die Attraktivität von Android von ausschlaggebender Bedeutung. Die Linux–Kernel–Komponente verleiht Android die notwendige Flexibilität zur Übertragung auf verschiedene Hardwareplattformen und Java wird aufgrund seiner Vorzüge als Programmiersprache von einer stetig wachsenden Zahl von Entwicklern genutzt. Andere mobile Plattformen haben nach Auffassung des Verfassers eine geringere Anerkennung, da sie diese Vorteile so nicht in sich vereinigen. In Abbildung 2.1 wird die Architektur von Android anhand der fünf Hauptkomponenten dargestellt, die dann näher erläutert werden.

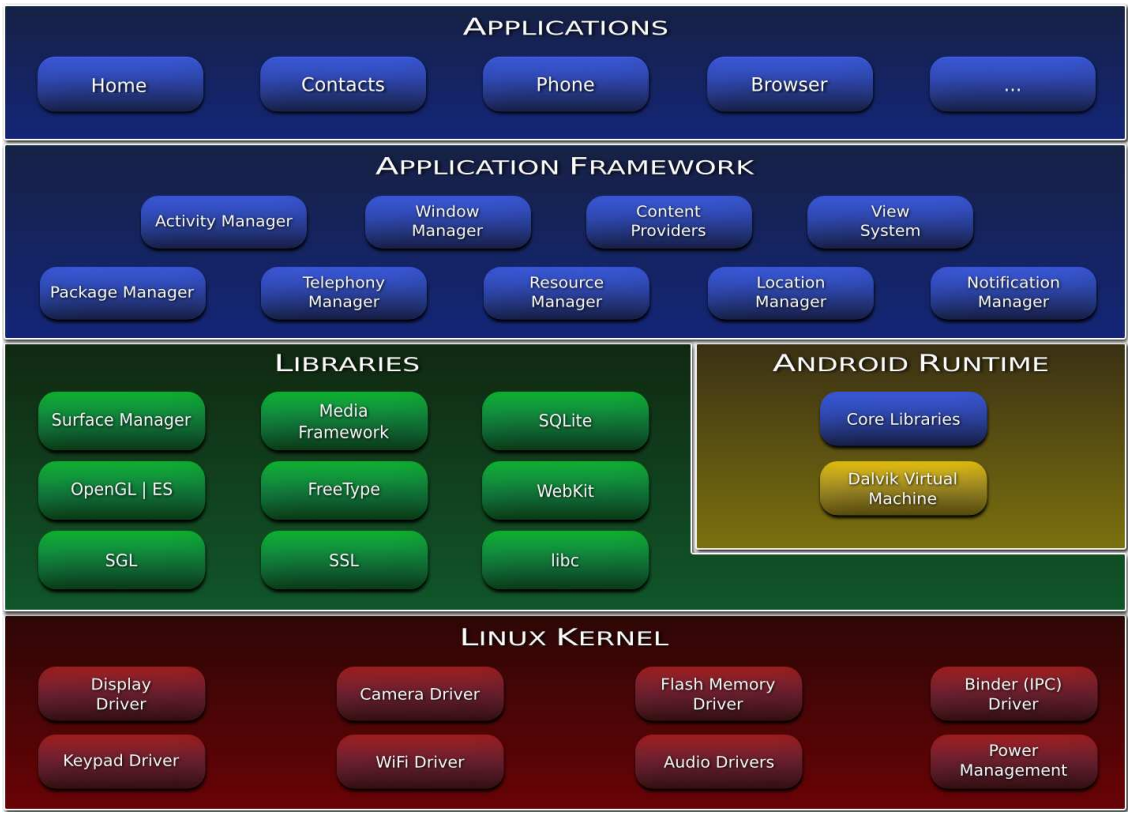

Abbildung 2.1: Systemarchitektur Android [3]

2.1.1 Applications

Android wird mit einigen Grundanwendungen ausgeliefert. Dazu gehören ein E–Mail–, eine SMS–, eine Kalender–, eine Karten–, eine Kontakte– und eine Browseranwendung. Alle Anwendungen wurden mit der Programmiersprache Java implementiert.

2.1.2 Application Framework

Das Anwendungsframework (Application Framework) stellt den Rahmen für eine einheitliche Anwendungsarchitektur bereit. Ziel ist es, Anwendungen nach einheitlichen Richtlinien zu entwickeln. Ein Beispiel hierfür ist, dass jede Anwendung ihre Fähigkeiten veröffentlichen und jede andere Anwendung diese Fähigkeiten dann bei Bedarf benutzen

3 `http://code.google.com/android/images/system-architecture.jpg`

kann. Die gleiche Vorgehensweise erlaubt es einem Entwickler, einzelne Komponenten durch neue oder eigene zu ersetzen. Die Einhaltung der Richtlinien vereinfacht somit die Integration und Wiederverwendung von Anwendungen unter Android zu vereinfachen.

Für die Entwicklung einer Android–Anwendung steht eine Sammlung von Diensten und Funktionen bereit, die im Folgenden genauer betrachtet werden (vgl. [Goo08b]).

- **Views** sind Ansichten, die es dem Anwender ermöglichen, mit der Anwendung zu agieren. Android beinhaltet eine reichhaltige Sammlung von erweiterbaren Ansichten, die dazu benutzt werden können, komplexere Ansichten zu erstellen. Zu den Grundansichten gehören Listen, Grids, Textboxen, Buttons.

- **Content Provider** bieten die Möglichkeit, auf Daten zuzugreifen oder diese für andere Anwendungen bereitzustellen. *Content Provider* stellen eine abstrakte Sicht auf die Daten dar. Wie ein *Content Provider* letztendlich Daten speichert, wird bei seiner Implementierung festgelegt (s. Punkt 2.3.4).

- Über den **Resource Manager** können externe Dateien, die keinen Programmcode darstellen und in die Anwendung einkompiliert worden sind, von dieser benutzt werden. Android unterstützt hierfür eine Vielzahl von unterschiedlichen Formaten. Diese werden in XML–Dateien, Bilddateien und Rohdaten eingeteilt.

- Der **Notification Manager** ermöglicht es Anwendungen, Nachrichten in der Statusbar anzuzeigen. Weiterhin kann an einer Nachricht ein *Intent* hinterlegt werden, sodass beim Auswählen dieser Nachricht Anwendungen regieren können.

- Der **Activity Manager** verwaltet den Lebenszyklus einer Anwendung. Er ermöglicht darüber hinaus eine Navigationsrückverfolgung.

2.1.3 Bibliotheken (*Libraries*)

Android beinhaltet eine Reihe von C/C++ Bibliotheken, die von verschiedenen Komponenten genutzt werden. Diese werden darüber hinaus auch den Entwicklern durch das Android–Anwendungsframework zur Verfügung gestellt. Die Basisbibliotheken sind:

- Die **System C Library** ist eine Implementierung der Standard C–Bibliothek durch die Berkeley Software Distribution (*BSD*). Sie wurde für Geräte auf Basis des Betriebssystems Linux optimiert.

- Die **Media Libraries** basieren auf dem Produkt OpenCORE der Firma Packet Video. Sie unterstützen die Wiedergabe und Aufnahme bekannter Musik–, Filmformate und statischer Bilder wie zum Beispiel MPEG4, H.264, MP3, AAC, AMR, JPG und PNG.

- Der **Surface Manager** verwaltet den Zugriff auf das Anzeigeuntersystem und ist für die nahtlose Vermischung von 2D und 3D Grafikschichten aus mehreren Anwendungen zuständig.

- Die **LibWebCore** ist eine moderne Web–Browser–Engine. Sie basiert auf der quelloffenen WebKit–Engine.

- Die **SGL** ist die zugrundeliegende 2D–Graphik–Engine.

- Die **3D libraries** sind eine Implementierung, die auf den *OpenGL ES 1.0 API* basieren. Die Bibliotheken verwenden entweder die 3D–Hardwarebeschleunigung, wenn diese vorhanden ist, oder den optimierten 3D–Software–Rasterizer.

- Der **FreeType** ist ein Bitmap– und Vektorschrift Renderer.

- Die **SQLite** ist eine leistungsstarke und kompakte relationale Datenbank–Engine, die für alle Anwendungen verfügbar ist.

2.1.4 Android Runtime

Android beinhaltet eine Reihe von Kernbibliotheken (Core Libraries), welche die meisten Funktionen zur Verfügung stellen, die auch in den Kernbibliotheken von Java zur Verfügung gestellt werden.

Jede Android–Anwendung läuft als eigener Prozess und mit ihrer eigenen Instanz der *Dalvik virtuellen Maschine* (*Dalvik*). *Dalvik* wurde mit dem Ziel entwickelt, auf einem Gerät mehrere virtuelle Maschinen (VM) effizient parallel ausführen zu können. *Dalvik* führt Dateien in dem für *Dalvik* entwickelten Format (.dex) aus. Dieses Format ist für einen minimalen Speicherverbrauch optimiert. *Dalvik* ist, wie die meisten eingesetzten ARM–Prozessoren, *registerbasiert* und nicht *stapelbasiert*. Hierdurch benötigt die VM weniger Zwischenschritte um den Bytecode auf dem Prozessor auszuführen. Die *Dalvik* führt Klassen aus, die mit einem Java Compiler kompiliert wurden und über das dx–Werkzeug in das .dex–Format umgewandelt worden sind. *Dalvik* nutzt die Funktionalität des darunter liegenden Linux–Kernels für das *threading* und die *Low–Level–Speicherverwaltung*.

2.1.5 Linux Kernel

Die Android Kernsystemdienste basieren hinsichtlich der Sicherheit, der Speicherverwaltung, der Prozessverwaltung, der Netzwerkschicht und des Treibermodells auf der Linux–Kernel–Version 2.6. Der Kernel dient zusätzlich als Abstraktionsschicht zwischen der Hardware– und den Softwareschichten.

2.2 Intent

Die folgenden Ausführungen gehen auf die zentrale Komponente von Android ein. Anhand dieser Komponente wird das Hauptziel verständlich, das mit der Entwicklung von Android verfolgt wurde. Ziel dieser Vorgehensweise ist es, eine bessere Sicht auf den Aufbau von Android–Anwendungen zu vermitteln. Der zentrale und immer wiederkehrende Begriff in der Android–Anwendungsentwicklung ist das *Intent*. („Understand the Intent, Understand Android.“ [Abl08, Seite 13]). Ein *Intent* ist das Bindeglied zwischen den vier Komponenten *Activity*, *Service*, *Content–Provider* und *Intent–Receiver* (s. Punkt 2.3). Es bewerkstelligt die Kommunikation zwischen diesen Komponenten. Mit dem *Intent* wird ein Vorhaben, eine Absicht beschrieben. Das könnte zum Beispiel das Öffnen eines Kontaktes oder einer Webseite sein. *Intents* erleichtern die Navigation auf innovative Art, denn alles ist nur einen „Klick“ entfernt. Die Objekte dieser „Klicks“ sind als *Uniform Resource Locator* (URL) oder alternativ auch als *Uniform Resource Identifier* (URI) bekannt. Der Einsatz von wirksamen URI ermöglicht den einfachen und schnellen Zugang zu Informationen, die der Benutzer ständig benötigt oder die er von anderen erhält. Die Art und Weise, wie ein mobiler Benutzer mit der Plattform interagiert, ist von entscheidender Bedeutung für deren wirtschaftlichen Erfolg. Plattformen, die auf mobilen Geräten lediglich eine klassische Desktopumgebung nachahmen, sind nur von einer kleinen Gruppe von erfahrenen Benutzern zufriedenstellend bedienbar. Tiefe Menüstrukturen, mehrere „Tabs“ und „Klicks“ werden von allen Nutzern mobiler Geräte nur ungern akzeptiert. Für eine Anwendung auf einem mobilen Gerät wird eine intuitive und einfache Nutzung gefordert. Die einfache Bedienung der Benutzeroberfläche (UI) eines beliebigen mobilen Endgerätemodells ist mitentscheidend für dessen Markterfolg.

UI können in diesem Zusammenhang auch einen Hinweis auf die Qualität des Datenzugriffsmodells einer Plattform geben. Nur wenn die Daten–Modelle gut strukturiert sind und der Datenzugriff effizient organisiert wird, sind gute Voraussetzungen gegeben, dem

Nutzer eine durchgängige, nach einfachen und einheitlichen Prinzipien arbeitende UI zur Verfügung zu stellen.

Der folgende Abschnitt geht näher auf den Bedienungs– und Auslösemechanismus ein, der durch die Kombination von *Intent* und *IntentFilter* verkörpert wird. Anwendungen für mobile Geräte und die darunter liegenden Basissysteme erfordern in noch stärkerem Maße eine möglichst einfache Bedienung, als man dies bereits von stationären Geräten gewohnt ist.

- Ein *Intent* enthält die Beschreibung einer Absicht oder die Anforderung eines Dienstes.
- Ein *IntentFilter* enthält Definitionen von Fähigkeiten und Interessen, die zur Unterstützung oder Umsetzung der Absicht erforderlich sein können. Er stellt die Beziehung zwischen dem *Intent* und der Anwendung her. Er kann allgemein, zum Datenteil oder zum Handlungsteil des *Intents* oder zu beiden spezifisch sein.

Das Handlungsattribut eines *Intent* ist typischer Weise ein Verb wie zum Beispiel *VIEW*, *PICK* oder *EDIT*. Es gibt eine Reihe von bereits eingebauten *Intent*–Handlungen, die Mitglieder der Klasse *Intent* sind. Es ist dem Anwendungsentwickler jedoch auch möglich, neue Handlungen zu erzeugen.

Dies soll an einem Beispiel veranschaulicht werden. Um eine Information anzusehen, würde eine Anwendung die folgende Handlung verwenden.

```
android.content.Intent.VIEW_ACTION
```

Die Datenkomponente eines *Intent* kann eine beliebige Information sein wie zum Beispiel ein Kontakteintrag, eine Webseite oder ein MP3–Ausschnitt. Die Daten werden mit einer URI beschrieben (s. Tabelle 2.1).

Art der Information	URI Daten
Kontakte nachschlagen	`content://contacts/people`
Adresse auf der Karte suchen	`geo:0,0?q=straße+adresse`
Webbrowser öffnen	`http://www.fh-brandenburg.de`

Tabelle 2.1: Beispiele für URI

Die Handlung `android.content.Intent.VIEW_ACTION` und die URI `content://contacts/people` würden auf der UI zu folgendem Ergebnis führen (Abb. 2.2):

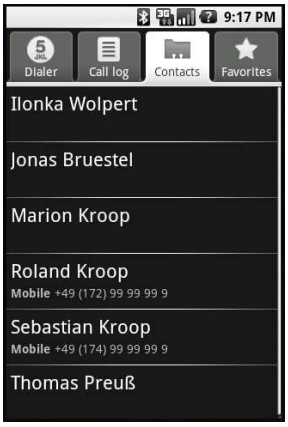

Abbildung 2.2: Ergebnis eines *Intents* auf der UI

Das Auflösen und Versenden von *Intents* geschieht zur Laufzeit und nicht beim Kompilieren der Anwendung. Dadurch ist die Möglichkeit gegeben, die Wirkung eines *Intents* durch Veränderung einer Anwendung auf dem Gerät zu beeinflussen. Hiermit können Aktualisierungen oder Erweiterungen vorgenommen werden.

Wird ein *Intent* abgeschickt, dann wertet das System die verfügbaren *Activities*, *Services* und registrierten *IntentReceiver* aus und leitet es dann zu dem am besten geeigneten Empfänger weiter. *Intents* werden zu allen Android–Anwendungen geleitet, die für diesen *IntentFilter* registriert sind.

Dieser Mechanismus wurde nach dem *Publish–Subscribe–Entwurfsmuster* (PSE) umgesetzt. Bei der vorliegenden Umsetzungsvariante des PSE registrieren sich die Teilnehmer (*Subscriber*) bei einem Vermittler, um über diesen dann später Nachrichten zu empfangen. Das Android–Framework realisiert die Registrierung durch das Einlesen und Verarbeiten der Datei `AndroidManifest.xml` zum Zeitpunkt der Installation einer Android–Anwendung. In dieser Datei werden Eigenschaften und damit auch die *Intents*, auf die Komponenten einer Android–Anwendung reagieren, aufgelistet.

Das Android–Framework (Vermittler im PSE) verwaltet alle Operationen und leitet Nachrichten zu den entsprechenden Empfängern weiter.

Der Initiator einer Nachricht (*Publisher*), kann einer der vier Applikationstypen sein. Er übermittelt Nachrichten an den Vermittler. Dieser leitet diese dann zu den entsprechenden *Subscribern* weiter (s. Abb. 2.3).

Durch die Benutzung des PSE ergeben sich einige Besonderheiten.

- Die Lebensdauer des *Publisher* und des *Subscriber* sind nicht von einander abhängig.
- Es kann mehrere Quellen für eine Nachricht geben.

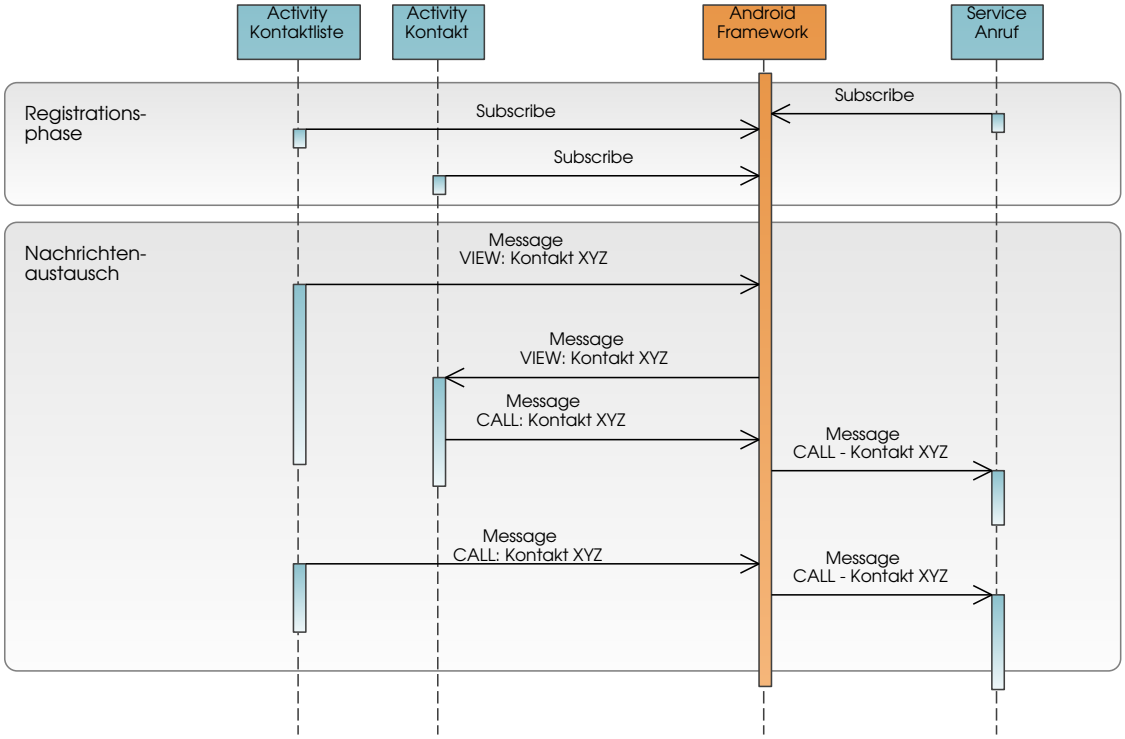

Abbildung 2.3: Schematische Darstellung des PSE anhand von Beispielen

Triviale Aufgaben bei der Nutzung eines mobilen Endgerätes sind zum Beispiel das Öffnen eines bestimmten Kontaktes, um diesen dann anzurufen, ihm eine Kurzmitteilung zu senden oder um die zu ihm gehörende E–Mail Adresse zu erfahren. Ein Benutzer könnte zum Beispiel auch eine bestimmte Information des Kontaktes XYZ nur einfach sehen wollen. In diesem Fall wäre die Handlung `VIEW` und die Daten sind die des gewünschten Kontaktes. Die Absicht wird durch Sendung eines *Intents* erfüllt, der die Handlung `VIEW` und eine URI enthält, die den Kontakt darstellt.

Ein Beispiel für eine URI, die mit der Handlung `android.content.Intent.VIEW_ACTION` benutzt werden kann, wäre:

- `content://contacts/people/xyz`

Um eine Liste von allen Kontakten zu erhalten, wird eine allgemeinere URI wie diese verwendet:

- `content://contacts/people`

In dem Codeausschnitt des Listing 2.1 wird gezeigt, wie man einen Kontakt auswählt.

```
1 Intent myintent = new Intent(Intent.PICK_ACTION,Uri.parse("content://contacts/
      /people"));
2 startActivity(myintent);
```

Listing 2.1: Kontakt auswählen

Das *Intent* in diesem Beispiel wird ausgewertet und an den am besten geeigneten Empfänger übergeben. Das in Android originär vorhandene *Activity* `com.google.android.phone.Dialer` ist in diesem Fall der Empfänger. Der beste Empfänger eines *Intent* kann wie hier ein vorhandenes *Activity*, ein *Activity* aus einer Android–Anwendung oder aus einer Drittanwendung sein.

In den Beispielen dieses Abschnitts wurde der Schwerpunkt auf *Intents* gelegt, welche die Benutzeroberfläche beeinflussen. Es gibt aber auch *Intents* die ereignisorientiert und nicht aufgabenorientiert sind, wie das im vorherigen Kontakteintragsbeispiel der Fall ist. Ein *Intent* kann ereignisorientiert benutzt werden, um zum Beispiel Anwendungen zu informieren, dass eine Textnachricht eingetroffen ist.

Für die Anwendungsentwicklung sind die beiden folgenden Verwendungsmöglichkeiten von *Intent* und *IntentFilter* erwähnenswert:

Anwendungen können durch Erzeugung eines *Intents* Funktionalität überlagern, die in anderen Anwendungen vorhanden ist. Hierdurch muss man Anwendungen nicht komplett neu schreiben. Ein großer Vorteil, der unter anderem beim Einsatz von *Intents* in dieser Art und Weise auftritt, ist zum Beispiel die effiziente Umsetzung einer einheitlichen Benutzeroberfläche. Dies ist für mobile Plattformen essentiell, da die Benutzer oft

technisch unversiert sind und kein Interesse daran haben, permanent neu zu lernen, wie man einen Kontakt auf dem Mobiltelefon sucht.

Um eine neue Funktionalität bereitzustellen, kann eine Android–Anwendung auch mit einem *IntentFilter* ausgestattet werden, der darauf hinweist, dass auf bereits vorhandene *Intents* zu reagieren ist.

2.3 Android–Komponenten

Dieser Abschnitt beschreibt die vier Grundbausteine einer Android–Anwendung (*Activity*, *Intent Receiver*, *Service*, *Content Provider*) und deren Beziehungen zum Prozessmodell von Android. Die folgenden Ausführungen zu den wichtigsten Klassen einer Android–Anwendung sollen deren Zusammenwirken im Grundsatz beschreiben. Das Verständnis dieser Zusammenhänge ist für die Interpretation und die Programmierung von Android–Anwendungen wichtig.

2.3.1 Activity

Eine Anwendung kann, muss aber keine Benutzeroberfläche haben. Wenn sie eine Benutzeroberfläche hat, dann wird sie eine oder mehrere *Activities* besitzen.

Der einfachste Weg eine Vorstellung von einem Android–*Activity* zu bekommen, ist die Assoziation eines Bildschirmzustandes mit einer Tätigkeit. In den meisten Fällen gibt es eine eins zu eins Beziehung zwischen einem *Activity* und der Benutzeroberfläche. Eine Android–Anwendung beinhaltet oft mehr als ein *Activity*. Jedes *Activity* stellt eine Benutzeroberfläche dar und reagiert auf System– und Benutzerereignisse. Das *Activity* verwendet hierfür eine oder mehrere *Views* um die aktuellen Bedienelemente anzuzeigen. Die *Activity*–Klasse wird durch Benutzerklassen erweitert (s. Listing 2.2).

```
1 package de.fhb.android.samples

3 import android.app.Activity;
4 import android.os.Bundle;

6 public class MyActivity extends Activity
7 {
8   @Override
9   public void onCreate(Bundle icicle)
10  {
11    super.onCreate(icicle);
12    setContentView(R.layout.main);
13  }
14 }
```

Listing 2.2: einfaches `Activity`

Erläuterungen zum Listing 2.2:

Zeile 3: Die Klasse *Activity* befindet sich in dem Java Paket `android.app`, welches in der Android Runtime enthalten ist. Die Android Runtime wird mit der `android.jar` Datei ausgeliefert.

Zeile 6: Die Klasse `MyActivity` ist abgeleitet von der Klasse `Activity`.

Zeile 12: Eine der Grundaufgaben eines *Activity* ist die Darstellung der Benutzeroberflächenelemente, die als *View* implementiert sind und in einer XML–Layoutdatei beschrieben werden.

Das Wechseln von einem *Activity* zu einem anderen wird durch die Methode `startActivity()` oder `startSubActivity()` bewerkstelligt. Die Methode `startSubActivity()` garantiert Synchronität zwischen dem Aufruf und dem Ergebnis. Als Argument wird den Methoden ein *Intent* übergeben.

Das *Activity* bildet eine sehr sichtbare Anwendungskomponente. Mit Unterstützung der *View*–Klasse ist das *Activity* die häufigste Art einer Android–Anwendung. Im nächsten Abschnitt wird der Grundbaustein *Service* erläutert. Dieser ist im Hintergrund aktiv und präsentiert keine Benutzeroberfläche.

2.3.2 Service

Wenn eine Anwendung einen langen Lebenszyklus besitzt, sollte diese als *Service* realisiert werden. Ein Werkzeug (Utility), das im Hintergrund Daten synchronisiert, wäre ein Beispiel hierfür.

Genau wie das *Activity* ist der *Service* eine Klasse, die von Android zur Laufzeit bereitgestellt wird und erweitert werden kann. Im Codeausschnitt des Listing 2.3 wird veranschaulicht, wie man periodisch eine Nachricht an das Android–Log senden kann.

```
1  package de.fhb.android.samples;

3  import android.app.Service;
4  import android.os.IBinder;
5  import android.util.Log;

7  public class MyService extends Service implements Runnable
8  {
9    public static final String tag = "MyService";
10   private int myiteration = 0;

12   @Override
13   protected void onCreate()
14   {
15     Thread mythread = new Thread (this);
16     mythread.start();
17   }

19   public void run()
20   {
21     while (true)
22     {
23       try
24       {
25         Log.i(tag,"MyService: " + myiteration++);
26         Thread.sleep(10000);
27       }
28       catch(Exception e)
29       {
30         Log.e(tag,e.getMessage());
31       }
32     }
33   }

35   @Override
36   public IBinder onBind(Intent intent)
37   {
38     // TODO Auto-generated method stub
39     return null;
40   }
41 }
```

Listing 2.3: Beispiel für einen *Service*

Erläuterungen zum Listing 2.3:

Zeile 3: Dieses Beispiel benötigt den Import des Paketes `android.app.Service`. Das Paket enthält die Klasse `Service`.

Zeile 5: Für den *logging*–Mechanismus in Android, der für das Debuggen sehr hilfreich sein kann, wird das Paket `android.util.Log` benötigt.

Zeile 7: Die Klasse `MyService` ist von `Service` abgeleitet. Um die Hauptaufgabe in einen separaten Thread auszulagern, implementiert die Klasse `Runnable`.

Zeile 13: Die `onCreate` Methode erlaubt es der Anwendung, Initialisierungsaufgaben durchzuführen.

Im nächsten Abschnitt wird der *IntentReceiver* näher betrachtet. Dies ist eine weitere Komponente von Android–Anwendungen. Sie ist dafür bestimmt, *Intents* zu verarbeiten.

2.3.3 Intent Receiver

Wenn eine Anwendung die Absicht hat, „globale“ Ereignisse zu empfangen und darauf zu reagieren, beispielsweise das Telefon klingelt oder eine Textnachricht trifft ein, so muss sie sich selbst als *IntentReceiver* registrieren. Eine Anwendung kann sich über die folgenden Wege registrieren, um auf *Intents* zu reagieren:

- Die Anwendung implementiert ein `<receiver>`–Element in der Datei `AndroidManfest.xml`, welches den *IntentReceiver*–Klassennamen beschreibt und dessen *IntentFilter* auflistet. Der *IntentFilter* beschreibt das *Intent*, auf das die Anwendung reagiert. Wenn der Empfänger in der Datei `AndroidManfest.xml` registriert ist, wird er durch das entsprechende Ereignis gestartet. Die Empfänger werden hierfür durch das Android–Framework verwaltet.

- Eine Anwendung registriert sich selbst zur Laufzeit über die Methode *registerReceiver* der Klasse *Context*.

- Android beinhaltet eine Reihe von eingebauten *IntentReceivern*, welche über die Methode *registerIntent* instanziiert und registriert werden können. Ein Beispiel hierfür ist der *PhoneStateIntentReceiver*. Dieser wird benutzt, um Telefonereignisse abzufangen wie zum Beispiel das Klingeln oder das Abheben des Hörers.

Genau wie der *Service* haben *IntentReceiver* keine Benutzeroberfläche.

Im Zusammenhang mit den *IntentReceivern* ist die Methode `Context.broadcastIntent` erwähnenswert. Diese Methode erlaubt es anderen zeitgleich laufenden Anwendungen, einen *Intent* zu senden. Auf diese Art und Weise können sich Anwendungen registrieren, die auf Ereignisse von anderen Anwendungen reagieren und dann ihrerseits Ereignisse an weitere Anwendungen versenden.

Ein *IntentReceiver* muss die abstrakte Methode `onReceiveIntent` implementieren, um auf eintreffende *Intents* zu reagieren. Die übergebenen Argumente für diese Methode sind ein *Context* und ein *Intent*. Die Methode selbst gibt ein `void` zurück. Es gibt aber verschiedene Methoden, um ein Ergebnis zurückzugeben. Eine Methode ist `setResult`, welche eine `integer`–Zahl für den Statuscode, einen `String` und ein `Bundle` für beliebige Objekte zurückgibt.

Das Codebeispiel aus dem Listing 2.4 ist ein Beispiel für einen *IntentReceiver*, der auf eine eintreffende Kurzmitteilung reagiert.

```
1  package de.fhb.android.samples;

3  import android.content.Context;
4  import android.content.Intent;
5  import android.content.IntentReceiver;
6  import android.util.Log;

8  public class MyIntentReceiver extends IntentReceiver
9  {
10   public static final String tag = "MyIntentReceiver";

12   @Override
13   public void onReceiveIntent(Context context, Intent intent)
14   {
15     Log.i(tag,"onReceiveIntent");

17     if (intent.getAction().equals("android.provider.Telephony.SMS_RECEIVED"))

18     {
19       Log.i(tag,"Found our Event!");
20     }
21   }
22 }
```

Listing 2.4: Beispiel für einen *IntentReceiver*

Erläuterungen zum Listing 2.4:

Zeile 8: Die Klasse `MyIntentReceiver` ist von der Klasse `IntentReceiver` abgeleitet. Dies ist eine unkomplizierte Art und Weise, einen *IntentReceiver* einzusetzen. Der Klassenname `MyIntentReceiver` wird in der Datei `AndroidManifest.xml` verwendet, wie im nächsten Codebeispiel aus Listing 2.5 zu erkennen ist.

Zeile 10: Die `tag`–Variable wird in Verbindung mit dem Log–Mechanismus verwendet. Nach diesem Tag kann dann im Debugger gefiltert werden. Dadurch wird das Debuggen vereinfacht.

Zeile 13: In der Methode `onReceiveIntent` findet die gesamte Arbeit des *IntentReceivers* statt. Der *IntentReceiver* wird benötigt, um diese Methode zu implementieren. Ein *IntentReceiver* kann mehrere *IntentFilter* registrieren und dadurch auch von mehreren *Intents* ausgelöst werden.

Zeile 17: Um das passende *Intent* zu behandeln, ist es wichtig, die Handlung der eingehenden *Intents* zu überprüfen.

Zeile 19: Ist das erwartete *Intent* eingetroffen, kann die erforderliche Funktionalität ausgeführt werden. Eine triviale Aufgabe wäre es zum Beispiel beim Erhalten einer Kurzmitteilung den Benutzer über den *Notification Manager* darüber zu informieren.

Damit der *IntentReceiver* auf dieses *Intent* reagieren kann, muss es in der Datei `AndroidManifest.xml` aufgeführt sein (s. Listing 2.5). Dieses Codebeispiel enthält alle notwendigen Elemente, um auf eine eintreffende Textnachricht zu reagieren.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3               package="de.fhb.android.samples">
4   <uses-permission android:name="android.permission.RECEIVE_SMS" />

6   <application android:icon="@drawable/icon">
7     <activity android:name="MyIntentReceiverActivity" android:label="@string/↯
          app_name">
8       <intent-filter>
9         <action android:name="android.intent.action.MAIN" />
10        <category android:name="android.intent.category.LAUNCHER" />
11      </intent-filter>
12    </activity>

14    <receiver android:name="MyIntentReceiver" >
15      <intent-filter>
16        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
17      </intent-filter>
18    </receiver>
19  </application>
20 </manifest>
```

Listing 2.5: Beispiel für eine `AndroidManifest.xml` mit `IntentReceiver`

Erläuterungen zu Listing 2.5:

Zeile 4: Für einige Aufgaben werden spezielle Privilegien benötigt. Um einer Anwendung die benötigte Berechtigung zu erteilen, muss dieses mit dem Tag `<uses-permission>` in der Datei `AndroidManifest.xml` angegeben werden.

Zeile 14: Das `<receiver>`–Tag beinhaltet den Klassennamen der Klasse, die den *IntentReceiver* implementiert. In diesem Beispiel ist es die Klasse `MyIntentReceiver` aus dem Paket `de.fhb.android.samples`.

Zeile 15: Der *IntentFilter* wird innerhalb des `<intent-filter>`–Tag definiert. Die Handlung, auf die reagiert werden soll, ist `android.provider.Telephony↯.SMS_RECEIVED`. Das SDK enthält eine Auflistung der verfügbaren Aktionen für die Standard–*Intents*. Darüber hinaus können Benutzeranwendungen eigene *Intents* verwenden und auf diese reagieren.

2.3.4 Content Provider

Der *Content Provider* stellt eine abstrakte Sicht auf Datenspeicher bereit. Auf diese können mehrere Anwendungen zugreifen. Weiterhin kann kontrolliert werden, in welcher Art und Weise auf die Daten zugegriffen wird.

Wenn eine Anwendung Daten verwaltet und diese Daten für weitere Anwendungen in der Umgebung von Android bereitgestellt werden sollen, dann sollte ein *ContentProvider* implementiert werden. Umgekehrt, wenn eine Anwendungskomponente (*Activity*, *Service* oder *IntentReceiver*) Daten einer anderen Anwendung benötigt, dann sollte der *ContentProvider* dieser Anwendung genutzt werden. Der *ContentProvider* implementiert Standardmethoden, um es einer Anwendung zu ermöglichen, auf einen Datenspeicher zuzugreifen. Der Zugriff kann benutzt werden, um Lese– oder Schreiboperationen auszuführen. Der *ContentProvider* kann Daten für ein *Activity* oder *Service* aus der gleichen Anwendung oder einer anderen Anwendung bereitstellen.

Ein *ContentProvider* kann jede verfügbare Form des Datenspeichers auf einem Android–Gerät benutzen. Dies können Dateien, SQLite–Datenbanken und speicherbasierte Hash–Maps sein, sofern für letztere die Persistenz der Daten nicht notwendig ist. Der *ContentProvider* ist im Wesentlichen eine Datenzugriffsschicht, um von den gespeicherten Daten mit dem Ziel zu abstrahieren, eine zentrale Speicherung und Abfrageroutinen in Einheit zu ermöglichen.

Der Content Provider ist die „Datenschicht“ für Android–Anwendungen. Er ist die empfohlene Art und Weise, um auf gespeichert Daten zuzugreifen oder um diese freizugeben.

Das direkte Freigeben von Dateien und Datenbanken ist auf Android nicht erwünscht. Darüber hinaus wird dies durch das Linux–Sicherheitssystem unterbunden, das Dateizugriffe von einem Anwendungsbereich in einen anderen verhindert, wenn die hierfür notwendigen Rechte nicht vorhanden sind.

Datentypen die in einem *ContentProvider* gespeichert werden, können zunächst allgemein bekannte Typen sein, wie das zum Beispiel bei `integer` oder `string` der Fall ist. Der *ContentProvider* kann aber auch Binärdaten verwalten wie zum Beispiel Bilddaten. Werden Binärdaten abgerufen, so wird empfohlen, einen `string` zurückzugeben, der den Dateinamen jener Datei darstellt, in der die Binärdaten enthält. Falls im Rahmen einer *ContentProvider*–Abfrage ein Dateiname zurückgegeben wird, sollte auf die Datei nicht direkt zugegriffen werden. Für den Zugriff auf Binärdaten stellt die Hilfsklasse *ContentResolver* eine Methode `openInputStream` bereit. Dieser Ansatz vermeidet Linux Prozess– /Sicherheitshürden und normalisiert alle Datenzugriffe durch den *ContentProvider*. Die Abbildung 2.4 zeigt die Beziehung zwischen dem *ContentProvider*, den Datenspeichern und den Clients.

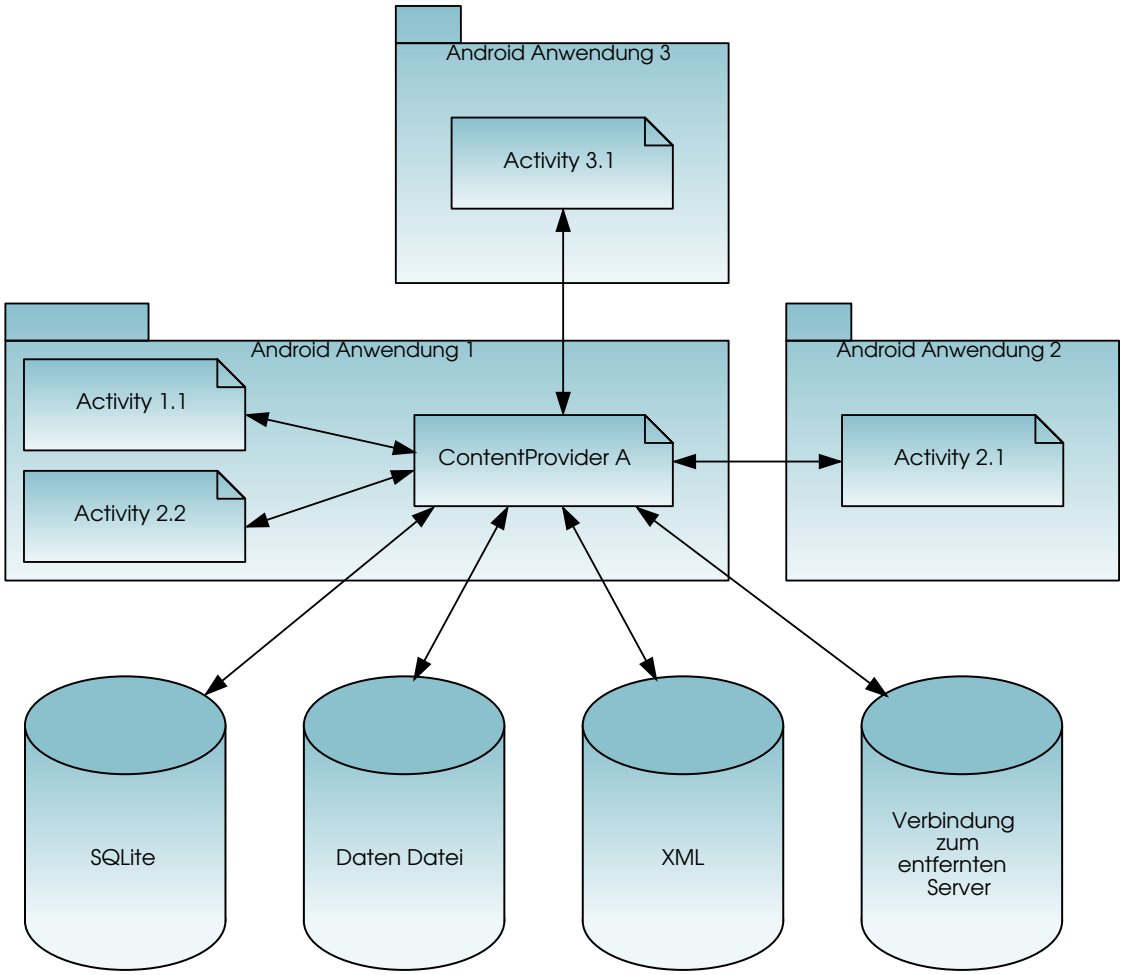

Abbildung 2.4: Beziehung zwischen Content Providern Datenspeichern und Clients

Auf die Daten eines *ContentProvider* wird über eine URI zugegriffen. Ein *ContentProvider* definiert diese als `public static final String`. Zum Beispiel könnte eine Anwendung einen Datenspeicher besitzen, in dem Wegstecken verwaltet werden sollen. Die URI für diesen *ContentProvider* könnte dann wie folgt aussehen:

- ```
  public static final Uri CONTENT_URI = ↯
  Uri.parse("content://de.fhb.android.location.tracking/tracks");
  ```

Auf diesem Niveau ist der Zugriff auf einen *ContentProvider* ähnlich der Benutzung einer *Structured Query Language* (SQL), wie man sie von anderen Plattformen kennt. Eine komplettes SQL–Statement wird jedoch nicht verwendet. Eine Abfrage mit den benötig-
  ```

ten Spalten wird dem *ContentProvider* übergeben. Optional können die `where`– und `order by`–Bedingungen mit übergeben werden. Die Parametersubstitution wird ebenfalls unterstützt. Ergebnisse werden in der `Cursor`–Klasse zurückgegeben.

Der nächste Abschnitt geht näher auf die `AndroidManifest.xml` Datei ein. In dieser werden globale Werte eines Android–Paketes festgelegt.

2.3.5 AndroidManifest.xml

In den vorhergehenden Abschnitten wurden die Standardelemente einer Android–Anwendung behandelt. An dieser Stelle muss noch einmal an einen wichtigen Aspekt hieraus erinnert werden: Eine Android–Anwendung kann ein oder mehrere *Activity*, *Service*, *IntentReceiver* und *ContentProvider* enthalten. Einige dieser Elemente veröffentlichen ihre *Intents* über den *IntentFilter* Mechanismus. Diese Einzelinformationen müssen jedoch zusammengefasst werden, um eine Android–Anwendung zu starten. Dies geschieht in der Datei `AndroidManifest.xml`.

Die `AndroidManifest.xml` Datei liegt im Wurzelverzeichnis einer Android–Anwendung und beinhaltet alle Aspekte einer spezifischen Anwendung und der *Intents*, die zur Entwicklungszeit festgelegt wurden. Das Codebeispiel aus dem Listing 2.6 ist eine sehr einfach `AndroidManifest.xml` Datei.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3                package="de.fhb.android.samples">
4      <application android:icon="@drawable/icon">
5          <activity android:name="MyActivity" android:label="@string/app_name">
6              <intent-filter>
7                  <action android:name="android.intent.action.MAIN" />
8                  <category android:name="android.intent.category.LAUNCHER" />
9              </intent-filter>
10         </activity>
11     </application>
12 </manifest>
```

Listing 2.6: Beispiel für eine einfache `AndroidManifest.xml`

Erläuterungen zu Listing 2.6:

Zeile 3: Das Element `Manifest` beinhaltet neben dem obligatorischen Namensraum den Java–Paketnamen, der diese Anwendung beinhaltet.

Zeile 5: Diese Anwendung beinhaltet ein *Activity* mit dem Klassennamen `MyActivity`. Das `@` Zeichen in der `@string`–Syntax wird benutzt, um die Information aus einer der Ressourcendateien zu referenzieren. In diesem Beispiel erhält das `label` Attribut seinen Inhalt aus der „`app_name`“ Text–Ressource.

Zeile 6: Dieses *Activity* beinhaltet eine *IntentFilter*–Definition. Der hier verwendete *IntentFilter* ist der in einer Android–Anwendungen am meisten benutzte. Das `action` Element `android.intent.action.MAIN` deutet auf den Einstiegspunkt der Anwendung hin. Das `category` Element `android.intent.category.LAUNCHER` platziert das *Activity* im Starterfenster (Abb. 2.5)

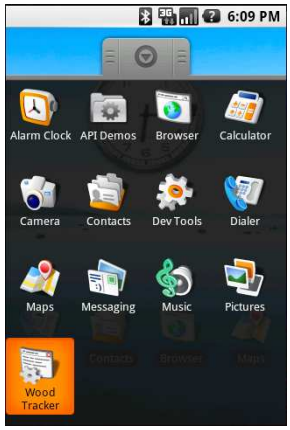

Abbildung 2.5: Anwendung im Starterfenster

Außer den Elementen in Listing 2.6, können weitere Tags der `AndroidManifest.xml` Datei hinzugefügt werden:

- Der `<service>`–Tag stellt einen *Service* dar. Als Attribute wird der Klassenname und seine Bezeichnung erwartet. Ein *Service* kann ein `<intent-filter>`–Tag beinhalten.

- Das `<receiver>`–Tag stellt einen `IntentReceiver` dar, der ein `<intent-filter>`–Tag beinhalten kann.

- Das `<uses-permission>`–Tag teilt Android mit, dass diese Anwendung einige Sicherheitsprivilegien benötigt. Wenn eine Anwendung lesenden Zugriff auf die Kontakte des Gerätes benötigt, wird folgender Tag in der Datei `AndroidManifest.xml` benötigt.

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

Der nächste Abschnitt geht auf die Beziehung zwischen einer Android–Anwendung und ihrer Laufzeitumgebung (Linux und Dalvik) ein.

2.3.6 Anwendungen und ihre Prozesse

Jede Android–Anwendung läuft in ihrem eigenen Linux–Prozess. Die Prozessverwaltung von Android unterliegt Linux und die Android–Anwendung läuft in einer Instanz der *Dalvik*. Um Ressourcen freizugeben, muss das Betriebssystem nicht prioritäre Anwendungen entladen oder sogar beenden. Hierfür trifft das System eine Auswahl anhand folgender Prioritäten:

- Sichtbare Anwendungen haben die höchste Priorität.
- Im Hintergrund aktive, vorübergehend pausierende Prozesse haben die nächst niedrigere Priorität, da sie demnächst wieder gestartet werden könnten.
- Die nächste niedrigere Priorität haben *Services*
- Prozesse die mit hoher Wahrscheinlichkeit beendet werden, sind jene die „schlafend“ oder „leer“ sind. Letztere wurden zuvor von einer Anwendung mit dem Ziel aktiviert, Komponenten einer Anwendung bei Bedarf schneller zu laden.

2.4 Hardwareressourcen

Speicher

Es gibt die Möglichkeit Daten, die sich nicht ändern, wie zum Beispiel Icons oder Hilfsdateien, in die Anwendung zu integrieren. Im Gerät steht ein kleiner Speicher für Datenbanken, Benutzerdateien oder für Daten bereit, die von der Anwendung empfangen wurden. Bei optionaler Verwendung einer Secure Digital Memory Card (SD Memory Card) hat der Nutzer auch dort die Möglichkeit des Zugriffs.

Netzwerk

Android–Geräte werden generell über ein Kommunikationsmedium internetfähig sein. Der Internetzugriff kann in jeder Schicht verwendet werden. Dies reicht von den Java Sockets bis hin zu einem Webbrowser, der in eine Anwendung integriert werden kann.

Multimedia

Android Geräte besitzen optional die Fähigkeit Ton und Video aufzuzeichnen und wiederzugeben. Ob diese Fähigkeit vorhanden ist, kann abgefragt werden.

Ortungsdienste

Einige Android–Geräte haben Zugriff auf Ortungsdienste wie zum Beispiel GPS oder einen Kompass.

Telefondienste

Typischerweise sind Android–Geräte primär Telefone. Hierdurch ist die Möglichkeit gegeben, Telefonate zu initiieren und Kurzmitteilungen zu senden und zu empfangen.

3 Entwurf

Im Punkt 2 wurden die Grundlagen für die Programmierung von Anwendungen auf Basis der Plattform Android ermittelt und dargestellt. Auf den dabei gewonnenen Kenntnissen aufbauend wurde dann die Anwendung *Woodtracker* (*Woodtracker*) konzipiert und prototypisch umgesetzt. Ziel war es, nach Abschluss dieser Arbeiten eine Evaluierung der Plattform vornehmen zu können.

Woodtracker soll es ermöglichen, zurückgelegte Wegstrecken aufzuzeichnen, um dann daraus später ein Wegenetz für die mobile Navigation erstellen zu können (s. Punkt 1.3).

Dafür soll zunächst die Erfüllung der folgenden Anforderungen prototypisch umgesetzt werden:

- Aufzeichnung zurückgelegter Wegstrecken

  Nach Starten der Anwendung soll das Gerät im Hintergrund zurückgelegte Wegstrecken anhand von periodisch erfassten Positionsdaten aufzeichnen.

- einfache Benutzung

  Beginn und Beendigung der Aufzeichnung sollen durch den Benutzer des mobilen Endgerätes auf einfache Art und Weise vorgenommen werden können.

- Darstellung des eigenen Standortes auf der UI

  Der eigene Standort soll dem Benutzer auf einer Karte anhand der aktuellen Position visuell dargestellt werden.

Unter der Bedingung, dass diese Anforderungen unter Android in sinnvoller Art und Weise umsetzbar sind, ist die Umsetzung weiterer Anforderungen zu einem späteren Zeitpunkt geplant:

- Darstellung bereits aufgezeichneter Strecken auf einer Karte
- Datenaustausch über eine REST–Schnittstelle zur weiteren Aufbereitung

Im folgenden Abschnitt wird die Architektur der Anwendung *Woodtracker* beschrieben.

3.1 Architektur

Die Abbildung 3.1 zeigt eine Übersicht der Komponenten des *Woodtracker* und seine Stellung im Schichtmodell des Android. Darüber hinaus sind die Beziehungen der Komponenten von *Woodtracker* zu der darunterliegenden Schicht des Android–Frameworks dargestellt. Aus Sicht der Anwendung gliedert sich die Architektur unter Vernachlässigung des Betriebssystems in die drei Schichten Hardware, Android–Framework und Anwendungskomponenten. Der *Woodtracker* ist Bestandteil der Schicht der Anwendungskomponenten. *Woodtracker* seinerseits unterteilt sich in die Komponenten Dienst, Persistenz und Client. Der Dienst ist für das Aufzeichnen der Standortdaten verantwortlich. Hierfür wird der *ContentProvider* (s. Punkt 2.3.4) aus dem Android–Framework benutzt. Die Implementierung des *ContentProvider* erfolgt in der Persistenzkomponente. Die Clientkomponente ist für die Steuerung des Dienstes verantwortlich und zeigt die aktuelle Wegstrecke an. Das Framework ist für die Kommunikation dieser Anwendungskomponenten untereinander unter Zuhilfenahme von *Intents* (s. Punkt 2.2) verantwortlich. Es realisiert in Zusammenarbeit mit dem Betriebssystem den Zugriff auf die Hardware.

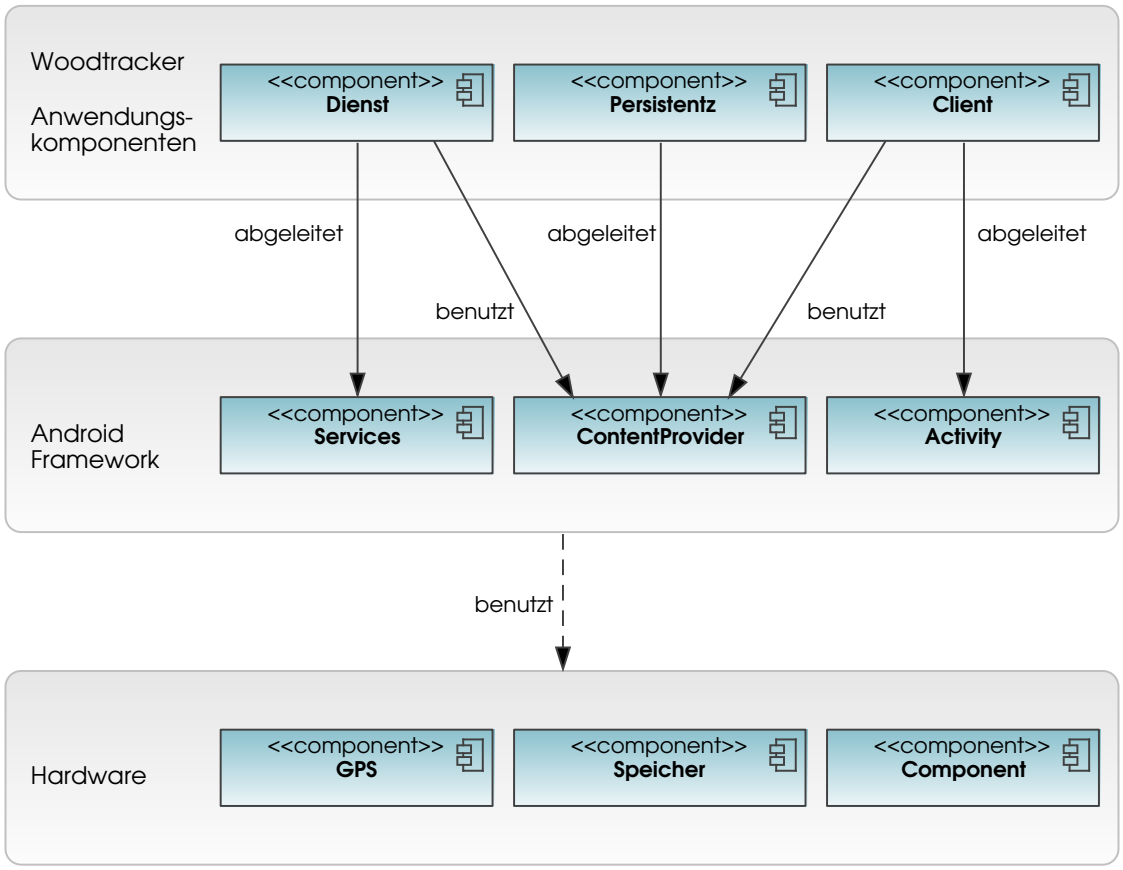

Abbildung 3.1: Anwendungsarchitektur Woodtracker

In den folgenden Abschnitten wird der Aufbau der Anwendungsschicht mit ihren Komponenten näher betrachtet.

3.1.1 Dienst

Abbildung 3.2 zeigt den Aufbau der Dienst–Komponente (*LocationTrackingService*). Ihre Aufgabe ist es, Standortdaten entgegenzunehmen und weiterzuverarbeiten, um diese dann zu speichern.

Die Teilkomponente *LocationTrackingManager* bezieht ihre Standortdaten über den *LocationManager*. Die Daten werden dann in einem weiteren Schritt an die Teilkomponente *TrackingRecorder* weitergeleitet. Dieser ist für die Aufbereitung der Daten zuständig und leitet diese seinerseits an den *ContentResolver* aus dem Android–Framework weiter, der

wiederum die Standortdaten dem ihm mitgeteilten *ContentProvider* übermittelt. Durch die Registrierung eines *IntentFilters* (s. Punkt 2.3.3) kann der Dienst über das Android–Framework angesprochen werden.

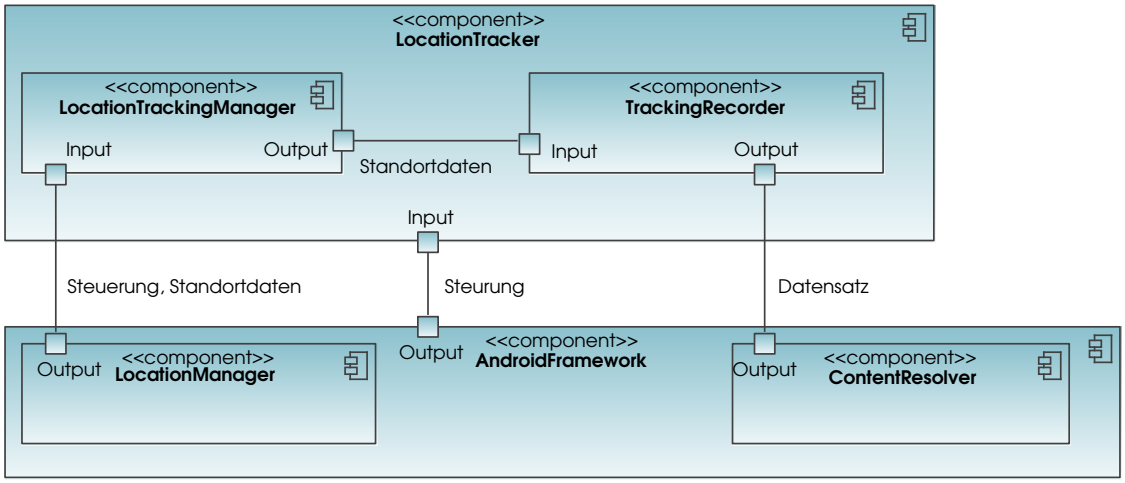

Abbildung 3.2: LocationTrackingService

3.1.2 Persistenz

Die persistente Speicherung und Bereitstellung der Wegstrecken für andere Android–Anwendungen auf dem selben Gerät wird durch die Komponente *TracksProvider* realisiert, die eine Erweiterung der Klasse *ContentProvider* ist (s. Punkt 2.3.4). Als Datenspeicher nutzt die Komponente eine SQLite–Datenbank. Abbildung 3.3 zeigt den schematischen Aufbau der Persistenz–Komponente des Woodtracker.

Durch die Benutzung eines *ContentResolver* (s. Punkt 2.3.4) können Android–Anwendungen über das Android–Framework Daten mit dem erweitertem *ContentProvider* austauschen.

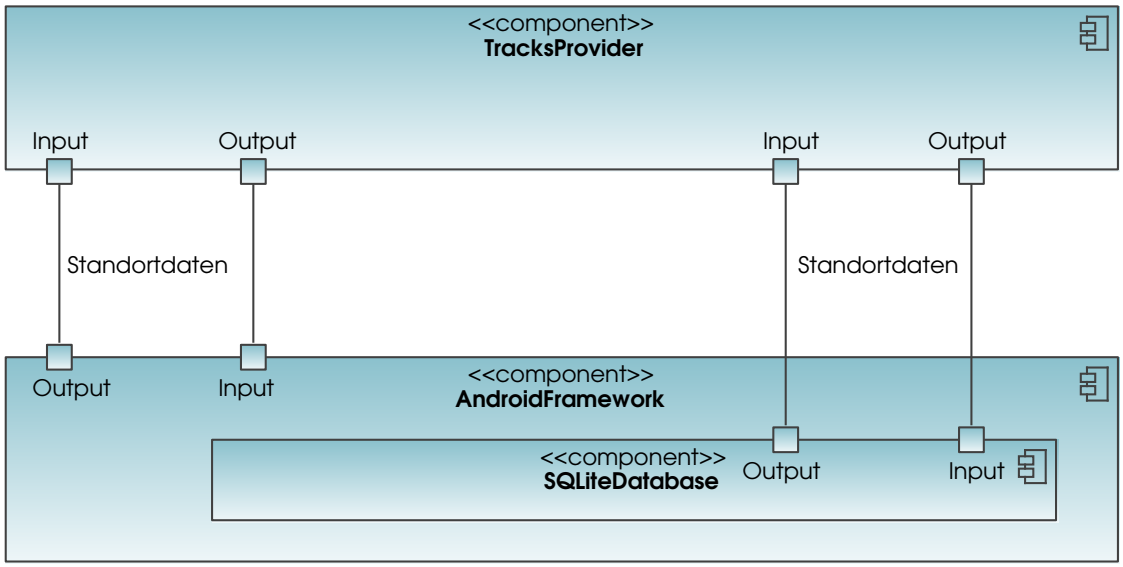

Abbildung 3.3: TracksProvider

3.1.3 Client

Die Komponente Client (*LocationTracker*) ist eine Erweiterung der *Activity*–Klasse (s. Punkt 2.3.1). Sie stellt das Bindeglied zwischen der Anwendung und dem Anwender dar und ist für das Zusammenführen sämtlicher Informationen und deren Darstellung auf dem Bildschirm verantwortlich. Darüber hinaus kann der Anwender über diese Komponente den Dienst steuern, der für die Aufzeichnung der Wegstrecken verantwortlich ist. Bereits gespeicherte Strecken können per *TracksProvider* durch den Nutzer ausgewählt und angezeigt werden. Abbildung 3.4 veranschaulicht den Aufbau der Komponente *LocationTracker*.

Eine wichtige Teilkomponente des *LocationTracker* ist die *RotateView*. Diese ist für das Zusammenführen der Bildoverlays *MapView*, *CurrentLocationOverlay* und *CurrentTrackOverlay* verantwortlich. Dabei liefert das *MapView–Overlay* das eigentliche Kartenmaterial. Quelle hierfür ist Google–Maps. Die Komponente *CurrentLocationOverlay* ist für die Darstellung des eigenen Standortes und *CurrentTrackOverlay* für die Darstellung der aktuell anzuzeigenden Wegstrecke zuständig. Ferner reagiert die Komponente *RotateView* auf Standortänderungen, die durch den *LocationManager* als Ereignis übermittelt werden. Zusätzlich ist die Komponente *RotateView* für die Aktualisierung des Bildschirmes verantwortlich.

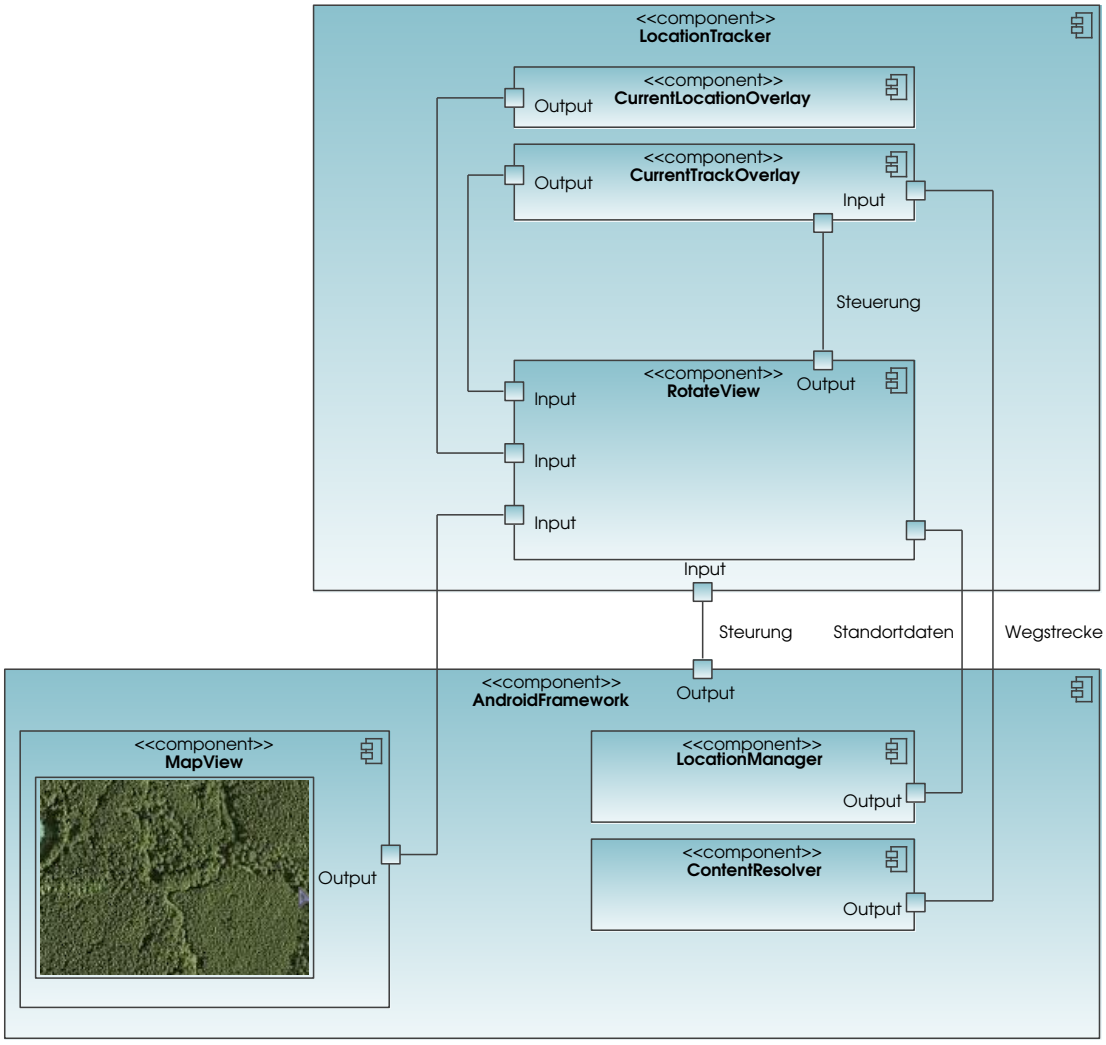

Abbildung 3.4: LocationTracker

4 Umsetzung

Dieses Kapitel beschreibt die prototypische Umsetzung eines Dienstes zur Aufzeichnung und Speicherung von Wegstrecken und einen Client zur Steuerung des Dienstes und zur Darstellung der aktuellen Position.

4.1 Entwicklungsumgebung

Für die Umsetzung wurde das Android Standard Developing Toolkit (Android SDK) der Version 1.0 (Release 1) verwendet. Als Entwicklungsumgebung diente Eclipse Version 3.4.1 mit der Erweiterung Android Development Tools (ADT) als Plugin. Eclipse und ADT ermöglichen es, eine Anwendung auf komfortable Art und Weise für Android auf einem Entwicklungs–PC zu erstellen und zu testen.

4.2 Dienst für die Aufzeichnung

Die Abbildung 4.1 zeigt das Paket `de.fhb.android.location.tracking.services`, dass die Pakete `impl` und `task` sowie die Klasse `LocationTrackingService` beinhaltet.

Die Klasse `LocationTrackingService` ist abgeleitet von der Klasse `android.app.Service` (s. Listing A.1, Zeile 17) des Android–Frameworks. Sie instanziiert die Klasse `LocationTrackingManager` (s. Listing A.1, Zeile 35). Sie umfasst die Ereignisse Initialisierung des Dienstes sowie Start und Beendigung der Wegstreckenaufzeichnung (s. Listing A.1, Zeile 28, Zeile 41, Zeile 49).

Das Paket `impl` enthält die Klasse `LocationTrackingManager` (s. Listing A.2. Diese ist für die Steuerung des Arbeitsablaufs der Wegstreckenaufzeichnung verantwortlich. Darüber hinaus implementiert sie die Schnittstelle `android.location.LocationListener`

(s. Listing A.2, Zeile 13) und ermöglicht damit die Entgegennahme der aktuellen geographischen Position, die sie dann an die Klasse `TrackRecorder` (s. Listing A.2, Zeile 79) weitergibt. Diese ist im Paket `task` (s. Listing A.3) enthalten und realisiert die Speicherung der Wegstrecken.

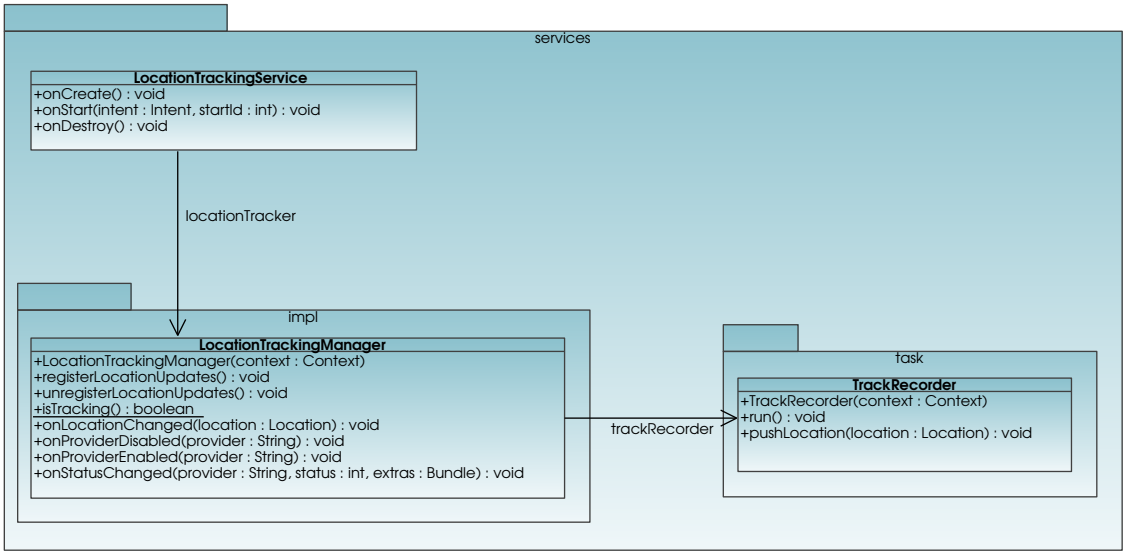

Abbildung 4.1: Paket `de.fhb.android.location.tracking.services`

4.3 Datenbereitstellung

Die Abbildung 4.2 zeigt das Paket `de.fhb.android.content`. Das Paket ermöglicht den Zugriff auf den Datenspeicher. Es beinhaltet die Klassen `TracksProvider` (s. Listing B.1), `TracksProviderBase` (s. Listing B.2) und `SQLiteDbHelper` (s. Listing B.3).

Die Klasse `TracksProviderBase` ist von der Klasse `android.content.ContentProvider` (s. Listing B.2, Zeile 11) abgeleitet. In ihr werden Eigenschaften definiert, die für die persistente Speicherung benötigt werden.

Die Klasse `SQLiteDbHelper` stellt eine Hilfsklasse für den Zugriff auf die SQLite–Datenbank dar. Hier werden die Aufgaben Datenbankerstellung und –aktualisierung erfüllt (s. Listing B.3, Zeile 57, Zeile 103). Weiterhin wird diese Klasse auch für die Kommunikation mit der Datenbank genutzt. Die Klasse `TracksProvider` ist von der Klasse `TracksProviderBase` abgeleitet (s. Listing B.1, Zeile 15) und stellt die Verwendung des

Android *ContentProvider* dar. Damit können die von Android vorgegebenen Methoden für den Datenbankzugriff verwendet und erweitert werden (s. Listing B.1, Zeile 32 und ff.).

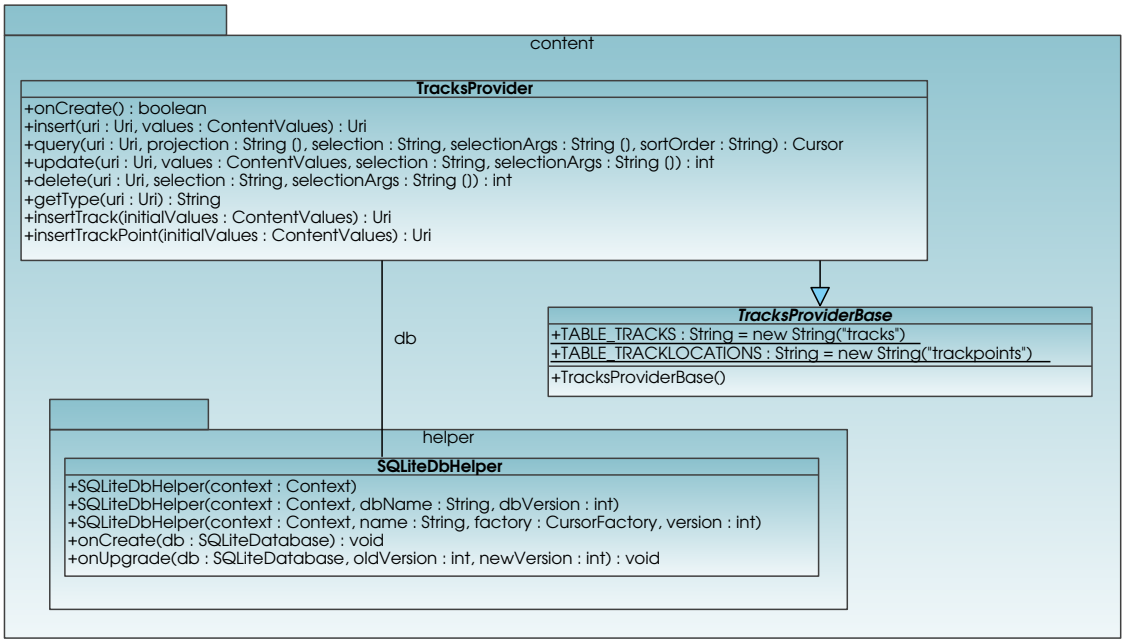

Abbildung 4.2: Paket `de.fhb.android.content`

4.4 Client–Anwendung

Die Abbildung 4.3 zeigt das Paket `de.fhb.android.location.tracking.clients`, das die Client–Komponente des `Woodtracker` darstellt. Das Paket enthält die Klassen `LocationTracker` (s. Listing C.1), `CurrentLocation` (s. Listing C.2), `RotateView` (s. Listing C.3) und `CurrentLocationOverlay` (s. Listing C.4).

Die Klasse `CurrentLocation` setzt das Singleton–Entwurfsmuster um und stellt die Standortdaten für das Paket `de.fhb.android.location.tracking.clients` bereit.

In der Klasse `RotateView` wird die Schnittstelle `android.location.LocationListener` implementiert (s. Listing C.3, Zeile 19), welche die Registrierung der aktuelle geographischen Position ermöglicht. Die Anzeige wird nach Norden ausgerichtet und auf den aktuellen Standort zentriert. Weiterhin speichert sie die Standortdaten in CurrentLocation (s. Listing C.3, Zeile 160).

Die Klasse `CurrentLocationOverlay` verarbeitet die aktuelle geographische Position aus `CurrentLocation`, indem sie die Anzeigeposition auf der Karte berechnet und an dieser Stelle ein Symbol darstellt.

Das Zusammenfügen der einzelnen Anzeige–Elemente wird durch die Klasse `LocationTracker` realisiert. Sie initialisiert die Klassen `RotateView` und `CurrentLocationOverlay` (s. Listing C.1, Zeile 41, Zeile 29).

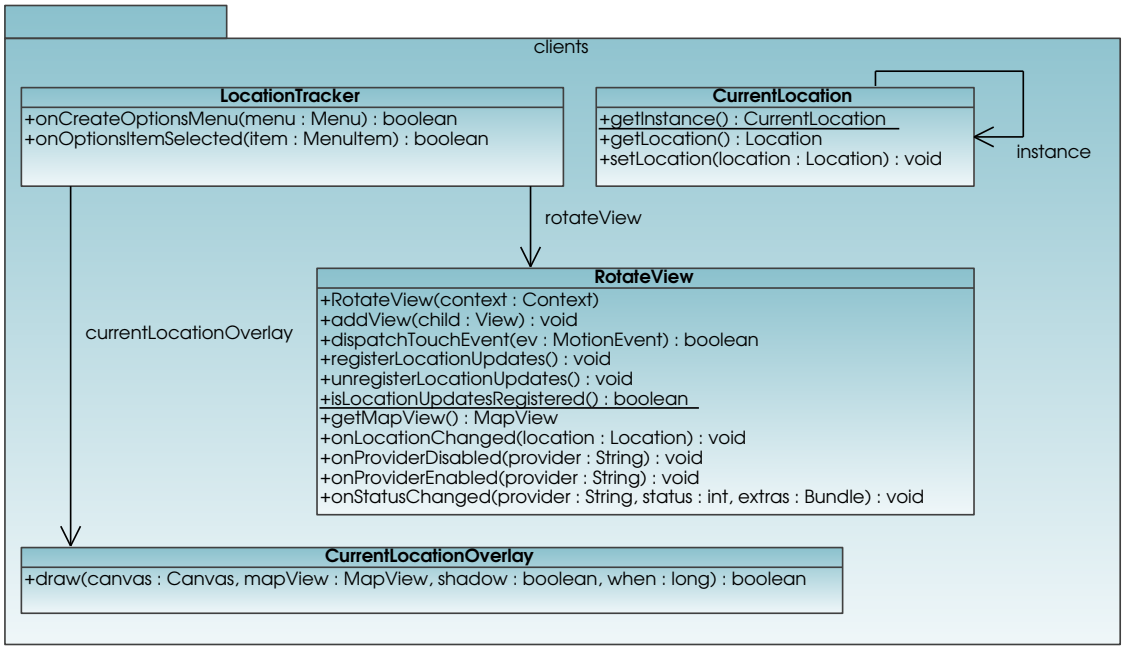

Abbildung 4.3: Paket `de.fhb.android.location.tracking.clients`

5 Zusammenfassung

Durch die Darstellung des Aufbau und der Wirkungsweise von Android wurden die potenziellen Möglichkeiten der Plattform sichtbar gemacht und gleichzeitig das Wissen zum Verständnis der Referenzanwendung bereitgestellt.

Die definierten Anforderungen (s. Punkt 3) für *Woodtracker* ließen sich aufgrund des Potenzials von Android problemlos umsetzen. Wegen des vorerst noch nicht verfügbaren Endgerätetyps konnte die Anwendung lediglich auf einer Emulation der Plattform Android getestet werden. Dies war mit einigen technischen Schwierigkeiten verbunden. Die Geschwindigkeit von Android und Anwendung war unter Emulationsbedingungen nicht zufriedenstellend. Android und Anwendung können einfach und intuitiv bedient werden.

Die Lauffähigkeit der Referenzanwendung unter Android konnte somit nachgewiesen werden. Nach Verfügbarkeit des mobilen Endgerätes für den praktischen Einsatz ist die Anwendung hierauf erneut zu testen. Die Benutzeroberfläche hatte die in sie gesetzten Erwartungen erfüllt. Die mit Android eingeschlagene Entwicklungsrichtung sollte weiter verfolgt werden. Die Übertragung der Anwendung auf andere Einsatzgebiete ist denkbar.

A Paket LocationTrackingService

A.1 Klasse LocationTrackingService

```
1  package de.fhb.android.location.tracking.services;
2
3  import android.app.Notification;
4  import android.app.NotificationManager;
5  import android.app.PendingIntent;
6  import android.app.Service;
7  import android.content.Context;
8  import android.content.Intent;
9  import android.os.IBinder;
10 import android.os.RemoteCallbackList;
11 import android.util.Log;
12 import android.widget.Toast;
13 import de.fhb.android.location.tracking.interfaces.IRemoteService;
14 import de.fhb.android.location.tracking.interfaces.IRemoteServiceCallback;
15 import de.fhb.android.location.tracking.services.impl.LocationTrackingManager;
       ;
16
17 public class LocationTrackingService extends Service
18 {
19     protected final String LOG_TAG = new String("LocationTrackingService");
20
21     private LocationTrackingManager locationTracker = null;
22
23     private final RemoteCallbackList<IRemoteServiceCallback> _mCallbacks =
           new RemoteCallbackList<IRemoteServiceCallback>();
24
25     NotificationManager notificationManager;
26
27     @Override
28     public void onCreate()
29     {
30         super.onCreate();
31
32         Log.i(this.LOG_TAG, "--> onCreate");
33
34         this.notificationManager = (NotificationManager) this.
               getSystemService(Context.NOTIFICATION_SERVICE);
35         this.locationTracker = new LocationTrackingManager(this);
```

```
37          this.showNotification();
38      }

40      @Override
41      public void onStart(Intent intent, int startId)
42      {
43          super.onStart(intent, startId);
44          Log.i(this.LOG_TAG, "--> onStart");
45          this.startTracking();
46      }

48      @Override
49      public void onDestroy()
50      {
51          Log.i(this.LOG_TAG, "--> onDestroy");

53          this.stopTracking();

55          this.notificationManager.cancel(R.string.remote_service_started);

57          Toast.makeText(this, R.string.remote_service_stopped, Toast.
                LENGTH_SHORT).show();

59  super.onDestroy();
60      }

62      @Override
63      public IBinder onBind(Intent intent)
64      {
65          Log.i(this.LOG_TAG, "--> onBind");

67          if (IRemoteService.class.getName().equals(intent.getAction()))
68              return this.mBinder;

70          return null;
71      }

73      private final IRemoteService.Stub mBinder = new IRemoteService.Stub()
74      {

76          public void registerCallback(IRemoteServiceCallback cb)
77          {
78              Log.i(LocationTrackingService.this.LOG_TAG, "--> registerCallback"
                    );

80              if (cb != null)
81                  LocationTrackingService.this._mCallbacks.register(cb);
82          }

84          public void unregisterCallback(IRemoteServiceCallback cb)
85          {
```

```
86              Log.i(LocationTrackingService.this.LOG_TAG, "--> ↯
                    unregisterCallback");

88              if (cb != null)
89                  LocationTrackingService.this._mCallbacks.unregister(cb);
90          }
91      };

93      private void showNotification()
94      {
95          Log.i(this.LOG_TAG, "--> showNotification");

97          CharSequence text = this.getText(R.string.remote_service_started);

99          Notification notification = new Notification(R.drawable.stat_sample, ↯
                text, System.currentTimeMillis());

101         PendingIntent contentIntent = PendingIntent.getActivity(this, 0, new ↯
                Intent("android.intent.action.MAIN"), 0);

103         notification.setLatestEventInfo(this, this.getText(R.string.↯
                remote_service_label), text, contentIntent);

105         this.notificationManager.notify(R.string.remote_service_started, ↯
                notification);
106     }

108     public void startTracking()
109     {
110         this.locationTracker.registerLocationUpdates();
111     }

113     public void stopTracking()
114     {
115         this.locationTracker.unregisterLocationUpdates();
116     }
117 }
```

Listing A.1: Klasse `LocationTrackingService`

A.2 Klasse LocationTrackingManager

```
1 package de.fhb.android.location.tracking.services.impl;

3 import java.util.List;

5 import de.fhb.android.location.tracking.services.task.TrackRecorder;

7 import android.content.Context;
```

```
8  import android.location.Location;
9  import android.location.LocationManager;
10 import android.os.Bundle;
11 import android.util.Log;

13 public class LocationTrackingManager implements android.location.
       LocationListener
14 {

16     private final static String LOG_TAG = new String("LocationTrackingManager
           ");

18     private final static long MINIMUM_DISTANCECHANGE_FOR_UPDATE = 0;
19     private final static long MINIMUM_TIME_BETWEEN_UPDATE = 500;

21     private final LocationManager locationManager;
22     private static TrackRecorder trackRecorder;

24     public LocationTrackingManager(Context context)
25     {
26         Log.i(LocationTrackingManager.LOG_TAG, "--> LocationTracking");

28         this.locationManager = (LocationManager) context.getSystemService(
               Context.LOCATION_SERVICE);
29         LocationTrackingManager.trackRecorder = new TrackRecorder(context);

31     }

33     private void setupForGPSAutoRefreshing()
34     {
35         Log.i(LocationTrackingManager.LOG_TAG, "--> setupForGPSAutoRefreshing
               ");

37         List<String> providers = this.locationManager.getAllProviders();

39         for (String provider : providers)
40             Log.i(LocationTrackingManager.LOG_TAG, "Provider found:" +
                   provider);

42         String strProvider = providers.get(0);

44         this.locationManager.requestLocationUpdates(strProvider,
45     LocationTrackingManager.MINIMUM_TIME_BETWEEN_UPDATE,
46     LocationTrackingManager.MINIMUM_DISTANCECHANGE_FOR_UPDATE, this);
47     }

49     public void registerLocationUpdates()
50     {
51         Log.i(LocationTrackingManager.LOG_TAG, "--> registerLocationUpdates");
               ;

53         if (!LocationTrackingManager.isTracking())
```

```
54          {
55              LocationTrackingManager.trackRecorder.start();
56              this.setupForGPSAutoRefreshing();
57          }
58      }

60      public void unregisterLocationUpdates()
61      {
62          if (LocationTrackingManager.isTracking())
63          {
64              this.locationManager.removeUpdates(this);
65              this.stopTrackRecorder();
66          }
67      }

69      public static boolean isTracking()
70      {
71          return LocationTrackingManager.trackRecorder.isAlive();
72      }

74      @Override
75      public void onLocationChanged(Location location)
76      {
77          Log.i(LocationTrackingManager.LOG_TAG, "--> onLocationChanged");

79          LocationTrackingManager.trackRecorder.pushLocation(location);

81      }

83      @Override
84      public void onProviderDisabled(String provider)
85      {
86          Log.i(LocationTrackingManager.LOG_TAG, "--> onProviderDisabled");

88      }

90      @Override
91      public void onProviderEnabled(String provider)
92      {
93          Log.i(LocationTrackingManager.LOG_TAG, "--> onProviderEnabled");

95      }

97      @Override
98      public void onStatusChanged(String provider, int status, Bundle extras)
99      {
100         Log.i(LocationTrackingManager.LOG_TAG, "--> onStatusChanged");

102     }

104     private void stopTrackRecorder()
105     {
```

```
106         Log.i(LocationTrackingManager.LOG_TAG, "Stopping TrackRecorder Thread
                ...");

108         while (LocationTrackingManager.trackRecorder.isAlive())
109         {
110             Log.i(LocationTrackingManager.LOG_TAG, "Still waiting...");

112             LocationTrackingManager.trackRecorder.interrupt();

114             while (LocationTrackingManager.trackRecorder.isInterrupted())
115             {
116                 try
117                 {
118                     Log.i(LocationTrackingManager.LOG_TAG, "Waiting for
                            TrackRecoder finishing...");
119                     LocationTrackingManager.trackRecorder.join();
120                 } catch (InterruptedException e)
121                 {
122                     Log.i(LocationTrackingManager.LOG_TAG, e.
                            getLocalizedMessage());
123                 }
124             }
125         }
126     }
127 }
```

Listing A.2: Klasse `LocationTrackingManager`

A.3 Klasse TrackRecorder

```
1  package de.fhb.android.location.tracking.services.task;

3  import java.util.LinkedList;

5  import de.fhb.android.location.tracking.interfaces.ITracks;
6  import de.fhb.android.location.tracking.interfaces.ITracks.ITrack;
7  import de.fhb.android.location.tracking.interfaces.ITracks.ITrackPoint;
8  import android.content.ContentResolver;
9  import android.content.ContentValues;
10 import android.content.Context;
11 import android.location.Location;
12 import android.net.Uri;
13 import android.util.Log;

15 public class TrackRecorder extends Thread
16 {
17     private final static String LOG_TAG = new String("TrackRecorder");

19     private final ContentResolver contentResolver;
```

```
20      private static Uri currentTrack = null;

22      private final LinkedList<Location> locations = new LinkedList<Location>()
            ;

24      private Location lastSavedLocation = null;

26      private long startTime = 0L;

28      public TrackRecorder(Context context)
29      {
30          this.contentResolver = context.getContentResolver();
31      }

33      @Override
34      public void run()
35      {
36          this.startNewTrack();

38          while (!this.isInterrupted() || !this.locations.isEmpty())
39          {
40              while (this.locations.size() > 1 || (this.isInterrupted() && this.
                    locations.size() > 0))
41              {
42                  if (this.lastSavedLocation == null)
43                  {
44                      Log.i(TrackRecorder.LOG_TAG, "FIRST Location save...");
45                      this.lastSavedLocation = this.saveLocation(this.locations.
                            removeFirst());
46                  } else
47                  {
48                      if ((this.isInterrupted() && this.locations.size() == 1)
49                              || !this.isLine(this.lastSavedLocation, this.
                                    locations.get(0), this.locations.get(1)))
50                      {
51                          Log.i(TrackRecorder.LOG_TAG, "Location save...");
52                          this.lastSavedLocation = this.saveLocation(this.
                                locations.removeFirst());
53                      } else
54                      {
55                          Log.i(TrackRecorder.LOG_TAG, "No need for Location save
                                ...");
56                          this.locations.removeFirst();
57                      }
58                  }
59              }

61              try
62              {
63                  Thread.sleep(10000);

65              } catch (InterruptedException e)
```

```
66              {
67                  this.interrupt();
68                  Log.i(TrackRecorder.LOG_TAG, "Stopping TrackRecorder...");
69              }
70          }
71      }

73      private void startNewTrack()
74      {
75          ContentValues values = new ContentValues();

77          this.startTime = System.currentTimeMillis();

79          Log.i(TrackRecorder.LOG_TAG, "--> startNewTrack");

81          values.put(ITrack.CREATED_DATE, this.startTime);
82          values.put(ITrack.MODIFIED_DATE, this.startTime);
83          values.put(ITrack.LABEL, "N/A");

85          TrackRecorder.currentTrack = this.contentResolver.insert(ITracks.↯
                CONTENT_URI, values);
86      }

88      public long getStartTime()
89      {
90          return this.startTime;
91      }

93      public void pushLocation(Location location)
94      {
95          if (!this.isInterrupted())
96              this.locations.add(location);
97          else
98              Log.i(TrackRecorder.LOG_TAG, "Location not added to List...");
99      }

101     private boolean isLine(Location pointA, Location pointB, Location pointC)
102     {
103         return false;
104     }

106     private Location saveLocation(Location location)
107     {
108         Log.i(TrackRecorder.LOG_TAG, "Sending Location to ContentProvider..."↯
                );

110         if (TrackRecorder.currentTrack != null)
111         {
112             ContentValues values = new ContentValues();

114             values.put(ITrackPoint.ACCURACY, location.getAccuracy());
115             values.put(ITrackPoint.ALTITUDE, location.getAltitude());
```

```
116             values.put(ITrackPoint.BEARING, location.getBearing());
117             values.put(ITrackPoint.LATITUDE, location.getLatitude());
118             values.put(ITrackPoint.LONGITUDE, location.getLongitude());
119             values.put(ITrackPoint.PROVIDER, location.getProvider());
120             values.put(ITrackPoint.SPEED, location.getSpeed());
121             values.put(ITrackPoint.TIME, location.getTime());

123             this.contentResolver.insert(TrackRecorder.currentTrack.buildUpon()
                    .appendPath("points").build(), values);

125             return location;
126         }

128         return null;
129     }
130 }
```

Listing A.3: Klasse `TrackRecorder`

B Paket TracksProvider

B.1 Klasse TracksProvider

```
1  package de.fhb.android.content;

3  import de.fhb.android.content.helper.SQLiteDbHelper;
4  import de.fhb.android.location.tracking.interfaces.ITracks;
5  import android.content.ContentUris;
6  import android.content.ContentValues;
7  import android.database.Cursor;
8  import android.database.SQLException;
9  import android.database.sqlite.SQLiteDatabase;
10 import android.database.sqlite.SQLiteQueryBuilder;
11 import android.net.Uri;
12 import android.provider.BaseColumns;
13 import android.util.Log;

15 public class TracksProvider extends TracksProviderBase
16 {
17     private static final String LOG_TAG = "TracksProvider";

19     private SQLiteDatabase db;

21     @Override
22     public boolean onCreate()
23     {
24         Log.i(TracksProvider.LOG_TAG, "--> onCreate");

26         this.db = (new SQLiteDbHelper(this.getContext())).getWritableDatabase
               ();

28         return (this.db == null) ? false : true;
29     }

31     @Override
32     public Uri insert(Uri uri, ContentValues values)
33     {
34         Log.i(TracksProvider.LOG_TAG, "--> insert");
35         int match = TracksProviderBase.URI_MATCHER.match(uri);
36         switch (match)
37         {
38             case TRACKS:
```

```
39              return this.insertTrack(values);

41          case TRACK_POINTS:
42              values.put(ITrackPoint.TRACK_ID, uri.getPathSegments().get(1))
                    ;
43              return this.insertTrackPoint(values);

45          default:
46              throw new IllegalArgumentException("Unknown URL " + uri);
47      }
48  }

50  @Override
51  public Cursor query(Uri uri, String[] projection, String selection,
        String[] selectionArgs, String sortOrder)
52  {
53      Log.i(TracksProvider.LOG_TAG, "--> query");

55      SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
56      StringBuilder where = new StringBuilder();

58      switch (TracksProviderBase.URI_MATCHER.match(uri))
59      {
60          case TRACKS:
61              qb.setTables(TracksProviderBase.TABLE_TRACKS);
62              qb.setProjectionMap(TracksProviderBase.
                    TRACK_LIST_PROJECTION_MAP);
63              break;

65          case TRACK_ID:
66              qb.setTables(TracksProviderBase.TABLE_TRACKS);

68              where.append(BaseColumns._ID).append("=").append(uri.
                    getPathSegments().get(1));
69              break;

71          case TRACK_POINTS:
72              qb.setTables(TracksProviderBase.TABLE_TRACKLOCATIONS);

74              where.append(ITrackPoint.TRACK_ID).append("=").append(uri.
                    getPathSegments().get(1));

76              qb.setProjectionMap(TracksProviderBase.
                    POINT_LIST_PROJECTION_MAP);
77              break;

79          case TRACK_POINT_ID:
80              qb.setTables(TracksProviderBase.TABLE_TRACKLOCATIONS);

82              where.append(ITrackPoint.TRACK_ID).append("=").append(uri.
                    getPathSegments().get(1));
83              where.append(" AND ");
```

```
84              where.append(BaseColumns._ID).append("=").append(uri.↯
                    getPathSegments().get(3));
85              break;

87          default:
88              throw new IllegalArgumentException("Unbekannte URL: " + uri);
89      }

91      if (where.length() >= 1)
92          qb.appendWhere(where);

94      Cursor c = qb.query(this.db, projection, selection, selectionArgs, ↯
            null, null, null);
95      c.setNotificationUri(this.getContext().getContentResolver(), uri);

97      return c;
98  }

100 @Override
101 public int update(Uri uri, ContentValues values, String selection, String↯
        [] selectionArgs)
102 {
103     Log.i(TracksProvider.LOG_TAG, "--> update");

105     return 0;
106 }

108 @Override
109 public int delete(Uri uri, String selection, String[] selectionArgs)
110 {
111     Log.i(TracksProvider.LOG_TAG, "--> delete");

113     return 0;
114 }

116 @Override
117 public String getType(Uri uri)
118 {
119     Log.i(TracksProvider.LOG_TAG, "--> getType");

121     int match = TracksProviderBase.URI_MATCHER.match(uri);

123     switch (match)
124     {
125         case TRACKS:
126             return "vnd.android.cursor.dir/vnd.de.fhb.android.track";

128         case TRACK_ID:
129             return "vnd.android.cursor.item/vnd.de.fhb.android.track";

131         case TRACK_POINTS:
```

```
132             return "vnd.android.cursor.dir/vnd.de.fhb.android.tracklocation";

134         case TRACK_POINT_ID:
135             return "vnd.android.cursor.item/vnd.de.fhb.android.tracklocation";

137         default:
138             throw new IllegalArgumentException("Unknown URL " + uri);
139     }
140 }

142 public Uri insertTrack(ContentValues initialValues)
143 {
144     long rowID;
145     ContentValues values;

147     if (initialValues != null)
148         values = new ContentValues(initialValues);
149     else
150         values = new ContentValues();

152     Long now = Long.valueOf(System.currentTimeMillis());

154     if (values.containsKey(ITrack.LABEL) == false)
155         values.put(ITrack.LABEL, new String("unbekannt"));

157     if (values.containsKey(ITrack.CREATED_DATE) == false)
158         values.put(ITrack.CREATED_DATE, now);

160     if (values.containsKey(ITrack.MODIFIED_DATE) == false)
161         values.put(ITrack.MODIFIED_DATE, now);

163     rowID = this.db.insert(TracksProviderBase.TABLE_TRACKS, null, values);

165     if (rowID > 0)
166     {
167         Uri uri = ContentUris.appendId(ITracks.CONTENT_URI.buildUpon(), rowID).build();
168         this.getContext().getContentResolver().notifyChange(uri, null);
169         return uri;
170     }
171     throw new SQLException("Failed to insert row");
172 }

174 public Uri insertTrackPoint(ContentValues initialValues)
175 {
176     long rowID;
177     ContentValues values;

179     if (initialValues != null)
```

```
180             values = new ContentValues(initialValues);
181         else
182             values = new ContentValues();
183
184         Long now = Long.valueOf(System.currentTimeMillis());
185
186         if (values.containsKey(ITrackPoint.ACCURACY) == false)
187             values.put(ITrackPoint.ACCURACY, 0L);
188
189         if (values.containsKey(ITrackPoint.ALTITUDE) == false)
190             values.put(ITrackPoint.ALTITUDE, 0L);
191
192         if (values.containsKey(ITrackPoint.BEARING) == false)
193             values.put(ITrackPoint.BEARING, 0L);
194
195         if (values.containsKey(ITrackPoint.LATITUDE) == false)
196             values.put(ITrackPoint.LATITUDE, 0L);
197
198         if (values.containsKey(ITrackPoint.LONGITUDE) == false)
199             values.put(ITrackPoint.LONGITUDE, 0L);
200
201         if (values.containsKey(ITrackPoint.PROVIDER) == false)
202             values.put(ITrackPoint.PROVIDER, new String("UNKNOWN"));
203
204         if (values.containsKey(ITrackPoint.SPEED) == false)
205             values.put(ITrackPoint.SPEED, 0L);
206
207         if (values.containsKey(ITrackPoint.TIME) == false)
208             values.put(ITrackPoint.TIME, 0L);
209
210         if (values.containsKey(ITrackPoint.TRACK_ID) == false)
211             values.put(ITrackPoint.TRACK_ID, 0L);
212
213         if (values.containsKey(ITrackPoint.MODIFIED_DATE) == false)
214             values.put(ITrackPoint.MODIFIED_DATE, now);
215
216         if (values.containsKey(ITrackPoint.CREATED_DATE) == false)
217             values.put(ITrackPoint.CREATED_DATE, now);
218
219         Log.i(TracksProvider.LOG_TAG, values.toString());
220
221         rowID = this.db.insert(TracksProviderBase.TABLE_TRACKLOCATIONS, null,↯
                values);
222
223         if (rowID > 0)
224         {
225             Uri uri = ContentUris.appendId(
226                                         ITracks.CONTENT_URI.buildUpon()
227                                                .appendEncodedPath(values.↯
                                                    getAsString(ITrackPoint.↯
                                                    TRACK_ID))
```

```
228                                                 .appendEncodedPath("points"), ↯
                                                        rowID).build();

230             this.getContext().getContentResolver().notifyChange(uri, null);
231             return uri;
232         }

234         throw new SQLException("Failed to insert row");
235     }
236 }
```

Listing B.1: Klasse `TracksProvider`

B.2 Klasse TracksProviderBase

```
1 package de.fhb.android.content;

3 import de.fhb.android.location.tracking.interfaces.*;
4 import java.util.HashMap;
5 import java.util.Map;

7 import android.content.ContentProvider;
8 import android.content.UriMatcher;
9 import android.provider.BaseColumns;

11 public abstract class TracksProviderBase extends ContentProvider implements ↯
       ITracks
12 {
13     protected static final int TRACKS = 1;

15     protected static final int TRACK_ID = 2;

17     protected static final int TRACK_POINTS = 3;

19     protected static final int TRACK_POINT_ID = 4;

21     protected static final UriMatcher URI_MATCHER = new UriMatcher(UriMatcher↯
           .NO_MATCH);

23     protected static Map<String, String> TRACK_LIST_PROJECTION_MAP = null;

25     protected static Map<String, String> POINT_LIST_PROJECTION_MAP = null;

27     public static final String TABLE_TRACKS = new String("tracks");

29     public static final String TABLE_TRACKLOCATIONS = new String("trackpoints↯
           ");

31     public TracksProviderBase()
```

```
32      {
33      }

35      static
36      {
37          TracksProviderBase.URI_MATCHER.addURI(ITracks.
                LOCATIONTRACKING_AUTHORITY, "tracks", TracksProviderBase.TRACKS);
38          TracksProviderBase.URI_MATCHER.addURI(ITracks.
                LOCATIONTRACKING_AUTHORITY, "tracks/#",
39                                              TracksProviderBase.TRACK_ID);
40          TracksProviderBase.URI_MATCHER.addURI(ITracks.
                LOCATIONTRACKING_AUTHORITY, "tracks/#/points",
41                                              TracksProviderBase.TRACK_POINTS);
42          TracksProviderBase.URI_MATCHER.addURI(ITracks.
                LOCATIONTRACKING_AUTHORITY, "tracks/#/points/#",
43                                              TracksProviderBase.TRACK_POINT_ID);

45          TracksProviderBase.TRACK_LIST_PROJECTION_MAP = new HashMap<String,
                String>();
46          TracksProviderBase.TRACK_LIST_PROJECTION_MAP.put(BaseColumns._ID,
                BaseColumns._ID);
47          TracksProviderBase.TRACK_LIST_PROJECTION_MAP.put(BaseColumns._COUNT,
                BaseColumns._COUNT);
48          TracksProviderBase.TRACK_LIST_PROJECTION_MAP.put(ITrack.LABEL,
                ITracks.ITrack.LABEL);
49          TracksProviderBase.TRACK_LIST_PROJECTION_MAP.put(ITrack.CREATED_DATE,
                 ITracks.ITrack.CREATED_DATE);
50          TracksProviderBase.TRACK_LIST_PROJECTION_MAP.put(ITrack.MODIFIED_DATE
                , ITracks.ITrack.MODIFIED_DATE);

52          TracksProviderBase.POINT_LIST_PROJECTION_MAP = new HashMap<String,
                String>();
53          TracksProviderBase.TRACK_LIST_PROJECTION_MAP.put(BaseColumns._ID,
                BaseColumns._ID);
54          TracksProviderBase.TRACK_LIST_PROJECTION_MAP.put(BaseColumns._COUNT,
                BaseColumns._COUNT);
55          TracksProviderBase.POINT_LIST_PROJECTION_MAP.put(ITrackPoint.TRACK_ID
                , ITrackPoint.TRACK_ID);
56          TracksProviderBase.POINT_LIST_PROJECTION_MAP.put(ITrackPoint.ACCURACY
                , ITrackPoint.ACCURACY);
57          TracksProviderBase.POINT_LIST_PROJECTION_MAP.put(ITrackPoint.ALTITUDE
                , ITrackPoint.ALTITUDE);
58          TracksProviderBase.POINT_LIST_PROJECTION_MAP.put(ITrackPoint.BEARING,
                 ITrackPoint.BEARING);
59          TracksProviderBase.POINT_LIST_PROJECTION_MAP.put(ITrackPoint.LATITUDE
                , ITrackPoint.LATITUDE);
60          TracksProviderBase.POINT_LIST_PROJECTION_MAP.put(ITrackPoint.
                LONGITUDE, ITrackPoint.LONGITUDE);
61          TracksProviderBase.POINT_LIST_PROJECTION_MAP.put(ITrackPoint.PROVIDER
                , ITrackPoint.PROVIDER);
62          TracksProviderBase.POINT_LIST_PROJECTION_MAP.put(ITrackPoint.SPEED,
                ITrackPoint.SPEED);
```

```
63          TracksProviderBase.POINT_LIST_PROJECTION_MAP.put(ITrackPoint.TIME, ↯
                ITrackPoint.TIME);
64          TracksProviderBase.POINT_LIST_PROJECTION_MAP.put(ITrackPoint.↯
                CREATED_DATE, ITrackPoint.CREATED_DATE);
65          TracksProviderBase.POINT_LIST_PROJECTION_MAP.put(ITrackPoint.↯
                MODIFIED_DATE, ITrackPoint.MODIFIED_DATE);
66      }
67  }
```

Listing B.2: Klasse `TracksProviderBase`

B.3 Klasse SQLiteDbHelper

```
1  package de.fhb.android.content.helper;

3  import android.content.Context;
4  import android.database.SQLException;
5  import android.database.sqlite.SQLiteDatabase;
6  import android.database.sqlite.SQLiteOpenHelper;
7  import android.database.sqlite.SQLiteDatabase.CursorFactory;
8  import android.provider.BaseColumns;
9  import android.util.Log;
10 import de.fhb.android.content.TracksProviderBase;
11 import de.fhb.android.location.tracking.interfaces.*;

13 public class SQLiteDbHelper extends SQLiteOpenHelper implements ITracks
14 {
15     private static final String LOG_TAG = new String("SQLiteDatabaseHelper");

17     private static final String DATABASE_NAME = new String("location_tracking↯
           .db");

19     private static final int DATABASE_VERSION = 1;

21     private static final Object[] trackDef = {TracksProviderBase.TABLE_TRACKS↯
           ,
22                                               BaseColumns._ID,
23                                               ITrack.LABEL,
24                                               ITrack.CREATED_DATE,
25                                               ITrack.MODIFIED_DATE};

27     private static final Object[] trackLocDef = {TracksProviderBase.↯
           TABLE_TRACKLOCATIONS,
28                                                  BaseColumns._ID,
29                                                  ITrackPoint.TRACK_ID,
30                                                  ITrackPoint.ACCURACY,
31                                                  ITrackPoint.ALTITUDE,
32                                                  ITrackPoint.BEARING,
33                                                  ITrackPoint.LATITUDE,
```

```
34                                              ITrackPoint.LONGITUDE,
35                                              ITrackPoint.PROVIDER,
36                                              ITrackPoint.SPEED,
37                                              ITrackPoint.TIME,
38                                              ITrackPoint.CREATED_DATE,
39                                              ITrackPoint.MODIFIED_DATE};

41      public SQLiteDbHelper(Context context)
42      {
43          this(context, SQLiteDbHelper.DATABASE_NAME, null, SQLiteDbHelper.
                DATABASE_VERSION);
44      }

46      public SQLiteDbHelper(Context context, String dbName, int dbVersion)
47      {
48          this(context, dbName, null, dbVersion);
49      }

51      public SQLiteDbHelper(Context context, String name, CursorFactory factory
            , int version)
52      {
53          super(context, name, factory, version);
54      }

56      @Override
57      public void onCreate(SQLiteDatabase db)
58      {
59          Log.i(SQLiteDbHelper.LOG_TAG, "Creating new Database...");

61          StringBuilder trackSqlStat = new StringBuilder("CREATE TABLE %1$s (")
62                  .append("%2$s INTEGER PRIMARY KEY AUTOINCREMENT,") // ID
63                  .append("%3$s TEXT,") // LABEL
64                  .append("%4$s INTEGER,") // CREATE_DATE
65                  .append("%5$s INTEGER") // MODIFIED_DATE
66                  .append(");");

68          StringBuilder trackLocSqlStat = new StringBuilder("CREATE TABLE %1$s
                (")
69                  .append("%2$s INTEGER PRIMARY KEY AUTOINCREMENT,") // ID
70                  .append("%3$s INTEGER,") // TRACK_ID
71                  .append("%4$s REAL,") // ACCURACY
72                  .append("%5$s REAL,") // ALTITUDE
73                  .append("%6$s REAL,") // BEARING
74                  .append("%7$s REAL,") // LATITUDE
75                  .append("%8$s REAL,") // LONGITUDE
76                  .append("%9$s TEXT,") // PROVIDER
77                  .append("%10$s REAL,") // SPEED
78                  .append("%11$s INTEGER,") // TIME
79                  .append("%12$s INTEGER,") // CREATE_DATE
80                  .append("%13$s INTEGER") // MODIFIED_DATE
81                  .append(");");
```

```
83          try
84          {
86              db.beginTransaction();
88              db.execSQL(String.format(trackSqlStat.toString(), SQLiteDbHelper.
                    trackDef));
89              db.execSQL(String.format(trackLocSqlStat.toString(), 
                    SQLiteDbHelper.trackLocDef));
91              db.setTransactionSuccessful();
93              db.endTransaction();
95          } catch (SQLException e)
96          {
97              Log.e(SQLiteDbHelper.LOG_TAG, e.getLocalizedMessage());
99          }
100     }
102     @Override
103     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
104     {
105         Log.i(SQLiteDbHelper.LOG_TAG, "Upgrading database from version " + 
                oldVersion + " to " + newVersion
106                 + ", which will destroy all old data");
108         StringBuilder crt = new StringBuilder("DROP TABLE IF EXISTS %1$s;");
110         db.beginTransaction();
112         db.execSQL(String.format(crt.toString(), SQLiteDbHelper.trackDef));
113         db.execSQL(String.format(crt.toString(), SQLiteDbHelper.trackLocDef))
                ;
115         db.setTransactionSuccessful();
117         db.endTransaction();
119         this.onCreate(db);
120     }
121 }
```

Listing B.3: Klasse `SQLiteDbHelper`

C Paket LocationTracker

C.1 Klasse LocationTracker

```
1  package de.fhb.android.location.tracking.clients;

3  import android.content.Context;
4  import android.content.Intent;
5  import android.os.Bundle;
6  import android.util.Log;
7  import android.view.Menu;
8  import android.view.MenuInflater;
9  import android.view.MenuItem;

11 import com.google.android.maps.MapActivity;
12 import com.google.android.maps.MapView;

14 import de.fhb.android.location.tracking.interfaces.IRemoteService;

16 public class LocationTracker extends MapActivity
17 {
18     // #### URL fr Generierung des API Key...
19     // http://code.google.com/android/maps-api-signup.html

21     // GPX-Files: http://tramper.co.nz/?view=browsegpxFiles

23     private static final String LOG_TAG = "LocationTracker";

25     private static final String APIKEY = new String("API_KEY");

27     private RotateView rotateView;
28     private MapView mapView;
29     private final CurrentLocationOverlay currentLocationOverlay = new ↯
           CurrentLocationOverlay();

31     @Override
32     protected void onCreate(Bundle savedInstanceState)
33     {
34         Log.d(LocationTracker.LOG_TAG, "--> onCreate");

36         this.bindService(new Intent(IRemoteService.class.getName()), ↯
               ServiceConnector.getInstance(),
37                         Context.BIND_AUTO_CREATE);
```

```
39          super.onCreate(savedInstanceState);

41          this.rotateView = new RotateView(this);

43          this.mapView = new MapView(this, LocationTracker.APIKEY);
44          this.mapView.getOverlays().add(this.currentLocationOverlay);

46          this.rotateView.addView(this.mapView);

48          this.setContentView(this.rotateView);

50          this.mapView.getController().setZoom(18);
51          this.mapView.setClickable(true);
52          this.mapView.setEnabled(true);
53      }

55      @Override
56      protected void onResume()
57      {
58          this.rotateView.registerLocationUpdates();
59          super.onResume();
60      }

62      @Override
63      protected void onStop()
64      {
65          this.rotateView.unregisterLocationUpdates();
66          super.onStop();
67      }

69      @Override
70      protected boolean isRouteDisplayed()
71      {
72          return false;
73      }

75      @Override
76      public boolean onCreateOptionsMenu(Menu menu)
77      {
78          MenuInflater inflater = this.getMenuInflater();
79          inflater.inflate(R.menu.option_main, menu);

81          return true;
82      }

84      @Override
85      public boolean onOptionsItemSelected(MenuItem item)
86      {
87          switch (item.getItemId())
88          {
89              case (R.id.record):
```

```
90                IRemoteService srv = ServiceConnector.getInstance().↯
                      getRemoteService();
91                break;

93            case (R.id.view):
94                break;

96            default:
97                break;
98        }
99        return true;
100   }

102   private void startRecording()
103   {
104   }

106   private void stopRecording()
107   {
108   }
109 }
```

Listing C.1: Klasse `LocationTracker`

C.2 Klasse CurrentLocation

```
1  package de.fhb.android.location.tracking.clients;

3  import android.location.Location;

5  public class CurrentLocation
6  {

8      static private CurrentLocation instance;

10     Location location;

12     private CurrentLocation()
13     {
14         this.initGlobals();
15     }

17     private void initGlobals()
18     {
19         this.location = null;
20     }

22     public static CurrentLocation getInstance()
23     {
```

```
25          if (CurrentLocation.instance == null)
26          {
27              synchronized (CurrentLocation.class)
28              {
29                  if (CurrentLocation.instance == null)
30                      CurrentLocation.instance = new CurrentLocation();
31              }
32          }

34          return CurrentLocation.instance;
35      }

37      public Location getLocation()
38      {
39          return this.location;
40      }

42      public void setLocation(Location location)
43      {
44          this.location = location;
45      }
46  }
```

Listing C.2: Klasse `CurrentLocation`

C.3 Klasse RotateView

```
1  package de.fhb.android.location.tracking.clients;

3  import java.util.List;

5  import com.google.android.maps.GeoPoint;
6  import com.google.android.maps.MapView;

8  import android.content.Context;
9  import android.graphics.Canvas;
10 import android.location.Location;
11 import android.location.LocationListener;
12 import android.location.LocationManager;
13 import android.os.Bundle;
14 import android.util.Log;
15 import android.view.MotionEvent;
16 import android.view.View;
17 import android.view.ViewGroup;

19 public class RotateView extends ViewGroup implements LocationListener
20 {
21     private static String LOG_TAG = new String("RotateView");
```

```
23      private final static long MINIMUM_DISTANCECHANGE_FOR_UPDATE = 1;
24      private final static long MINIMUM_TIME_BETWEEN_UPDATE = 2000;

26      private final LocationManager locationManager;
27      private static boolean LocationUpdatesRegistered;

29      private static final float SQ2 = 1.414213562373095f;
30      private final SmoothCanvas canvas = new SmoothCanvas();
31      private float heading = 0;

33      private int mapViewIdx = -1;

35      public RotateView(Context context)
36      {
37          super(context);

39          this.locationManager = (LocationManager) context.getSystemService(
                Context.LOCATION_SERVICE);
40      }

42      @Override
43      protected void dispatchDraw(Canvas canvas)
44      {
45          canvas.save(Canvas.MATRIX_SAVE_FLAG);
46          canvas.rotate(-this.heading, this.getWidth() * 0.5f, this.getHeight()
                * 0.5f);
47          this.canvas.setDelegate(canvas);
48          super.dispatchDraw(this.canvas);
49          canvas.restore();
50      }

52      @Override
53      protected void onLayout(boolean changed, int l, int t, int r, int b)
54      {
55          final int width = this.getWidth();
56          final int height = this.getHeight();
57          final int count = this.getChildCount();
58          for (int i = 0; i < count; i++)
59          {
60              final View view = this.getChildAt(i);
61              final int childWidth = view.getMeasuredWidth();
62              final int childHeight = view.getMeasuredHeight();
63              final int childLeft = (width - childWidth) / 2;
64              final int childTop = (height - childHeight) / 2;
65              view.layout(childLeft, childTop, childLeft + childWidth, childTop
                    + childHeight);
66          }
67      }

69      @Override
70      protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec)
```

```
71      {
72          int w = View.getDefaultSize(this.getSuggestedMinimumWidth(), ↯
                widthMeasureSpec);
73          int h = View.getDefaultSize(this.getSuggestedMinimumHeight(), ↯
                heightMeasureSpec);
74          int sizeSpec;
75          if (w > h)
76          {
77              sizeSpec = MeasureSpec.makeMeasureSpec((int) (w * RotateView.SQ2),↯
                    MeasureSpec.EXACTLY);
78          } else
79          {
80              sizeSpec = MeasureSpec.makeMeasureSpec((int) (h * RotateView.SQ2),↯
                    MeasureSpec.EXACTLY);
81          }
82          final int count = this.getChildCount();
83          for (int i = 0; i < count; i++)
84          {
85              this.getChildAt(i).measure(sizeSpec, sizeSpec);
86          }
87          super.onMeasure(widthMeasureSpec, heightMeasureSpec);
88      }

90      @Override
91      public void addView(View child)
92      {
93          Log.i(RotateView.LOG_TAG, "--> add ViewGroupChild");
94          super.addView(child);

96          if (((MapView) child) != null)
97              this.mapViewIdx = (super.getChildCount() - 1);
98      }

100     @Override
101     public boolean dispatchTouchEvent(MotionEvent ev)
102     {
103         return super.dispatchTouchEvent(ev);
104     }

106     private GeoPoint getPoint(double lat, double lon)
107     {
108         return (new GeoPoint((int) (lat * 1000000.0), (int) (lon * 1000000.0)↯
                ));
109     }

111     private void setupForGPSAutoRefreshing()
112     {
113         Log.i(RotateView.LOG_TAG, "--> setupForGPSAutoRefreshing");

115         List<String> providers = this.locationManager.getAllProviders();

117         for (String provider : providers)
```

```
118             Log.i(RotateView.LOG_TAG, "Provider found:" + provider);

120         String strProvider = providers.get(0);

122         this.locationManager.requestLocationUpdates(strProvider, RotateView.
                MINIMUM_TIME_BETWEEN_UPDATE,
123                                                 RotateView.
                                                        MINIMUM_DISTANCECHANGE_FOR_UPDATE
                                                        , this);
124     }

126     public void registerLocationUpdates()
127     {
128         Log.i(RotateView.LOG_TAG, "--> registerLocationUpdates");

130         if (!RotateView.isLocationUpdatesRegistered())
131         {
132             RotateView.LocationUpdatesRegistered = true;
133             this.setupForGPSAutoRefreshing();
134         }
135     }

137     public void unregisterLocationUpdates()
138     {
139         Log.i(RotateView.LOG_TAG, "--> unregisterLocationUpdates");
140         if (RotateView.isLocationUpdatesRegistered())
141         {
142             this.locationManager.removeUpdates(this);
143             RotateView.LocationUpdatesRegistered = false;
144         }
145     }

147     public static boolean isLocationUpdatesRegistered()
148     {
149         return RotateView.LocationUpdatesRegistered;
150     }

152     public MapView getMapView()
153     {
154         return (MapView) this.getChildAt(this.mapViewIdx);
155     }

157     public void onLocationChanged(Location location)
158     {

160         CurrentLocation.getInstance().setLocation(location);

162         Log.d(RotateView.LOG_TAG, "--> Location (Bea: " + location.getBearing
                () + "/ Lat: " + location.getLatitude()
163                 + " / Lon: " + location.getLongitude() + ")");
164         if (location.getTime() == -1000)
165             this.heading = location.getBearing();
```

```
167         this.getMapView().getController().setCenter(this.getPoint(location.↯
                getLatitude(), location.getLongitude()));

169         this.invalidate();
170     }

172     public void onProviderDisabled(String provider)
173     {

175     }

177     public void onProviderEnabled(String provider)
178     {

180     }

182     public void onStatusChanged(String provider, int status, Bundle extras)
183     {

185     }
186 }
```

Listing C.3: Klasse `RotateView`

C.4 Klasse CurrentLocationOverlay

```
1 package de.fhb.android.location.tracking.clients;

3 import android.graphics.Canvas;
4 import android.graphics.Paint;
5 import android.graphics.Point;
6 import android.graphics.RectF;
7 import android.graphics.Paint.Style;

9 import com.google.android.maps.GeoPoint;
10 import com.google.android.maps.MapView;
11 import com.google.android.maps.Overlay;

13 public class CurrentLocationOverlay extends Overlay
14 {
15     @Override
16     public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long ↯
           when)
17     {
18         super.draw(canvas, mapView, shadow);

20         if (CurrentLocation.getInstance().getLocation() == null)
21             return true;
```

```
23          Paint paint = new Paint();
24          paint.setTextSize(14);

26          Double lat = CurrentLocation.getInstance().getLocation().getLatitude
                () * 1E6;
27          Double lng = CurrentLocation.getInstance().getLocation().getLongitude
                () * 1E6;
28          GeoPoint point = new GeoPoint(lat.intValue(), lng.intValue());

30          Point myScreenCoords = new Point();
31          mapView.getProjection().toPixels(point, myScreenCoords);

33          RectF oval = new RectF(myScreenCoords.x - 7, myScreenCoords.y + 7,
                myScreenCoords.x + 7, myScreenCoords.y - 7);

35          paint.setStyle(Style.FILL);
36          paint.setARGB(255, 80, 150, 30);

38          paint.setARGB(80, 156, 192, 36);
39          paint.setStrokeWidth(1);

41          canvas.drawOval(oval, paint);


44          paint.setARGB(255, 0, 0, 0);
45          paint.setStyle(Style.STROKE);
46          canvas.drawCircle(myScreenCoords.x, myScreenCoords.y, 7, paint);

48          return true;
49      }
50  }
```

Listing C.4: Klasse `CurrentLocationOverlay`

C.5 Klasse CurrentLocationOverlay

```
1  package de.fhb.android.location.tracking.clients;

3  import android.graphics.Canvas;
4  import android.graphics.Paint;
5  import android.graphics.Point;
6  import android.graphics.RectF;
7  import android.graphics.Paint.Style;

9  import com.google.android.maps.GeoPoint;
10 import com.google.android.maps.MapView;
11 import com.google.android.maps.Overlay;
```

```
13 public class CurrentLocationOverlay extends Overlay
14 {
15     @Override
16     public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long ↯
           when)
17     {
18         super.draw(canvas, mapView, shadow);

20         if (CurrentLocation.getInstance().getLocation() == null)
21             return true;

23         Paint paint = new Paint();
24         paint.setTextSize(14);

26         Double lat = CurrentLocation.getInstance().getLocation().getLatitude↯
               () * 1E6;
27         Double lng = CurrentLocation.getInstance().getLocation().getLongitude↯
               () * 1E6;
28         GeoPoint point = new GeoPoint(lat.intValue(), lng.intValue());

30         Point myScreenCoords = new Point();
31         mapView.getProjection().toPixels(point, myScreenCoords);

33         RectF oval = new RectF(myScreenCoords.x - 7, myScreenCoords.y + 7, ↯
               myScreenCoords.x + 7, myScreenCoords.y - 7);

35         paint.setStyle(Style.FILL);
36         paint.setARGB(255, 80, 150, 30);

38         paint.setARGB(80, 156, 192, 36);
39         paint.setStrokeWidth(1);

41         canvas.drawOval(oval, paint);


44         paint.setARGB(255, 0, 0, 0);
45         paint.setStyle(Style.STROKE);
46         canvas.drawCircle(myScreenCoords.x, myScreenCoords.y, 7, paint);

48         return true;
49     }
50 }
```

Listing C.5: Klasse `CurrentLocationOverlay`

Abbildungsverzeichnis

Listingverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Abl08] ABLESON, W. Frank; COLLINS, Charlie; SEN, Robi und COOPER, Robert: *Unlocking Android A Developer's Guide*, Manning Publications Co., manning early access program Aufl. (März 2008) (Seite 9)

[Bor08] BORT, Dave: Android is now available as open source, http://source.android.com/posts/opensource (2008), (HTML) (Seite 2)

[Bro08] BROCKMAN, Joshua: Google Is Calling. Will You Answer?, http://www.npr.org/templates/story/story.php?storyId=94982690 (2008), (HTML) (Seite 2)

[Cla07] CLABURN, Thomas: Google's Secret Patent Portfolio Predicts gPhone, Information Week, http://www.informationweek.com/news/showArticle.jhtml?articleID=201807587&cid=nl_IWK_daily (2007), (HTML) (Seite 1)

[Cox07] COX, John: Why Google's phone won't kill Apple's iPhone, Network World 2. New York Times, http://www.networkworld.com/news/2007/100807-google-gphone-iphone.html (2007), (HTML) (Seite 1)

[Elg05] ELGIN, Ben: Google Buys Android for Its Mobile Arsenal, Business Week, http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm (2005), (HTML) (Seite 1)

[Goo08a] GOOGLE: Development Tools, http://code.google.com/intl/de-DE/android/intro/tools.html (2008), (HTML) (Seite 2)

[Goo08b] GOOGLE: What is Android?, http://code.google.com/intl/de-DE/android/what-is-android.html (2008), (HTML) (Seite 7)

[Ihl07] IHLENFELD, Jens: Google zeigt Handy-Plattform Android samt SDK, http://www.golem.de/0711/55956.html (2007), (HTML) (Seite 1)

[McK06] MCKAY, Martha: Can iPhone become your phone?; Linksys introduces versatile line for cordless service. *The Record (North Jersey Media Group)* (2006):S. L9 (Seite 1)

[Ope07] OPEN HANDSET ALLIANCE: Industry Leaders Announce Open Platform for Mobile Devices, http://www.openhandsetalliance.com/press_110507.html (2007), presse Veröffentlichung (Seite 1)

[Ric08] RICKER, Thomas: HTC Dream FCC approved, Android clear for launch?, http://www.engadget.com/2008/08/18/htc-dream-fcc-approved-android-clear-for-launch/ (2008), (HTML) (Seite 2)

[Sha07] SHARMA, Amol: Google Plans Search Service for Mobile Content, http://online.wsj.com/public/article/SB118461672269867869-vFHnSZtA5io6FL9LUX_ddqWQ49I_20070815.html (2007), (HTML) (Seite 1)

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die von mir eingereichte Bachelorarbeit selbstständig verfasst, ausschließlich die angegebenen Hilfsmittel benutzt und sowohl wörtliche, als auch sinngemäße entlehnte Stellen als solche kenntlich gemacht habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Potsdam, den 02.02.2009 Sebastian Kroop