Universität Hamburg
MIN-Fakultät
Department Informatik
AB Wissenschaftliches Rechnen
Wintersemester 2009/10

Seminar „Android: Plattform für mobile Geräte“

Schnittstellen

Christian Baumann
5959075
7. April 2010

Inhaltsverzeichnis

1 Einleitung

Am 21. Oktober 2008 veröffentlichte die Open Handheld Alliance das Handy-Betriebssystem Android. Es bietet eine Reihe von vorhandenen Bibliotheken und Unterstützung von diversen Komponenten (z. B.: GPS, WLAN und Sensoren).

Android baut auf dem Linux-Kernel 2.6 auf und steht somit unter der GPL. Dies bedeutet, Android ist quelloffen und als freie Software erhältlich. Die Laufzeitumgebung basiert auf einer der Java VM ähnlichen Virtual Machine, der Dalvik Virtual Machine. Diese erlaubt es Entwicklern, auf bestehende Entwicklungswerkzeuge für Java zurückzugreifen. Intern nutzt die Dalvik VM allerdings auch in C und C++ geschriebene Bibliotheken, um so einen Geschwindigkeitsvorteil zu erzielen.

Die Nutzung der Dalvik VM ermöglicht es, Programme für Android komplett in Java zu schreiben, die dann auf dem Android lauffähig sind.

Zusätzlich ist die gesamte Struktur des Systems stark modular aufgebaut. Dadurch können eigene Anwendungen jederzeit die von Android mitgelieferten Anwendungen ersetzen. Dies ist eine Neuheit auf dem Markt der Handy-Betriebsysteme, da es zwar schon möglich war, eigene Software auf dem Handy auszuführen, aber die meisten vom Hersteller vorgegebenen Anwendungen wie SMS-Versand und Wählsystem sich nicht ändern ließen. Allerdings setzt die Nutzung dieses betriebsnahen Environments klar definierte und dokumentierte Schnittstellen voraus.

2 Überblick über die Schnittstellen von Android

Android baut auf Java auf, bringt aber auch eigene Klassen mit. Ingesamt bietet Android 1448 Javaklassen an, von denen 511 Android-spezifisch sind. Das hat natürlich auch Auswirkungen auf die Schnittstellen.

2.1 Welche Schnittstellen gibt es?

Durch seinen modularen Aufbau benötigt Android natürlich diverse Schnittstellen. Diese lassen sich grob in fünf Kategorien unterteilen:

1. Grafik-Schnittstellen

   Diese Schnittstellen umfassen alle Schnittstellen, die auf die Grafik zugreifen. Android unterstützt sowohl 2D, als auch 3D. Während die 2D Bibliothek ein reichhaltiges Angebot an zweidimensionalen Formen bietet und auch die Einbindung von Bildern ermöglicht, bietet Android für 3D die OpenGL API.

2. Eingabe-Schnittstellen

   Hier sind alle Schnittstellen gemeint, die in irgendeiner Form die Eingabe vom Benutzer aufgreifen. Dies beinhaltet die Eingabe via Tastatur, Touch-Screen oder die Bewegungssensoren, sowie GPS.

3. Telefon-Schnittstellen

   Die Telefon-Schnittstellen sind zuständig für die Abwicklung der Telefonie. Dies beinhaltet den Aufbau einer Wählverbindung, Annahme sowie Ablehnung einer eingehenden Verbindung und die Unterbrechung einer bestehenden Verbindung.

4. Speicher-Schnittstellen

   Android bietet die Unterstützung von SD-Karten, USB-Massenspeichern, internem Speicher, sowie eine interne Datenstruktur in Form von SQLite. Um darauf zuzugreifen, bietet das Betriebssystem eine Vielzahl von Bibliotheken an.

5. Netzwerk-Schnittstellen

   Heutige Smartphones unterstützen eine Vielzahl von Netzwerk-Verbindungen (WLAN, UMTS, Bluetooth, GPRS). Android bietet Schnittstellen für alle diese Verbindungen an, sowei einige Besonderheiten, auf die ich später noch eingehen werde.

Insgesamt verfügt Android über 394 Schnittstellen. In meiner Ausarbeitung möchte ich mich auf eine Auswahl beschränken. Ich habe mich für eine Schnittstelle aus dem Bereich Speicher sowie eine grobe Übersicht über die Netzwerk-Schnittstellen entschieden, von denen ich drei näher darstellen werde.

Die Auswahl habe ich getroffen, da diese Schnittstellen eine Besonderheit von Android sind.

2.2 Besonderheiten von Schnittstellen

Android ist speziell für Smartphones entwickelt worden. Diese bringen eine Reihe von Besonderheiten mit.

Hier sei zum einen die komplette Untertützung der Telefonie genannt. Schließlich möchte man mit dem Handy auch gelegentlich telefonieren.

Aber auch bei den Netzwerk-Schnittstellen gibt es Besonderheiten. Da das Gerät mobil ist, kann es zum Beispiel passieren, dass der Nutzer, während er das WLAN benutzt, aus dem Bereich des Hotspots herausläuft. Natürlich müssen das Gerät und die Software darauf reagieren können. Es kann dadurch auch passieren, dass regelmäßig die Art der Verbindung wechselt, da das Handy eine Reihe verschiedener Netzwerkverbindungen unterstützt. Somit muss es eine Möglichkeit geben, dass die Software darauf auch Zugriff hat. Desweiteren hat das Gerät nur eine begrenzte Energieversorgung und muss deshalb sparsam mit den Ressourcen umgehen.

Auch beim Speicher müssen einige Besonderheiten berücksichtigt werden. Zum einen ist der Speicher des Gerätes begrenzt und nur in geringem Umfang nachrüstbar. Wenn der Speicher erweitert wird, zum Beispiel durch eine SD-Karte, muss damit gerechnet werden, dass die Karte auch entfernt wird und der Datenbestand damit nicht mehr zur Verfügung steht. Zu guter Letzt muss damit gerechnet werden, dass durch die Möglichkeit, externe Software auch in systemnahen Bereichen zu nutzen, auch schädliche Software ausgeführt werden kann. Deshalb muss verhindert werden, das Software auf Bereiche zugreifen kann, auf die sie nicht zugreifen soll.

Einige dieser Besonderheiten existieren auch auf anderen Geräten. Allerdings benutzen diese Geräte häufig ein vom Hersteller mitgeliefertes Betriebssystem. Dadurch kann es passieren, das ein Programm auf einem Nokia-Gerät, nicht aber auf einem Sony-Ericsson-Gerät, funktioniert. Läuft ein Programm aber auf dem einen Android-Gerät, läuft es auch auf jedem anderen mit derselben Android-Version.

3 Speicher-Schnittstelle: Content Provider

Android hat einige Besonderheiten im Bereich der Speichernutzung. Um aber weiter eine problemlose und einfache Nutzung zu ermöglichen, gibt es den Content Provider.

3.1 Warum benötigt man einen Content Provider?

Eine Besonderheit von Android liegt in dessen Sicherheitstruktur begründet. So laufen alle Programme in einer eigenen Sandbox. Dies bedeutet, die Programme laufen unabhängig voneinander. Auch gibt es keine Common-Storage-Area. Es ist also nicht möglich, dass ein Programm auf die Daten eines anderen Programms zugreift.

Häufig möchte man aber gerade auf Daten oder Dateien anderer Programme zugreifen können. Zum Beispiel möchte man auf ein Bild zugreifen, das im Anhang einer E-Mail mitgeschickt wurde. Oder man möchte den Text aus einer SMS in einem eigenen Textverarbeitungsprogramm nutzen. Aus diesem Grund liefert Android ein eigenes Schnittstellenkonzept mit: Den Content Provider.

Dieser ermöglicht es, auf die Dateien und Daten anderer Programme zuzugreifen und diese zu verändern. Ein Entwickler hat damit zwei Möglichkeiten. Zum einen kann er die vorgegebenen Content Provider nutzen, oder aber er erstellt eigene.

Da diese Schnittstellen genutzt werden müssen, können später auch andere Programme auf diese Provider zugreifen. Dies unterstützt den modularen Aufbau von Android.

Ein Beispiel:

Ein Programm bekommt Daten im XML-Format über das Netzwerk und formatiert diese dann zu verständlichen Textpaketen. Diese können dann per Content Provider von einem anderen Programm abgerufen und weiter verarbeitet werden, in diesem Fall ein Chat-Programm.

Soll das Programm nun erweitert werden, um zum Beispiel Bilder darüber zu empfangen, so ist dies möglich, ohne die Netzwerkanbindung oder das Chatprogramm zu verändern.

3.2 Funktionsweise

Der Content Provider kann direkt angesprochen werden, dann können die Daten aber nur gelesen werden. Alternativ kann mittels Content Resolver auf die entsprechenden Daten zugegriffen werden. Dann können diese auch verändert werden. Das sieht dann so aus:

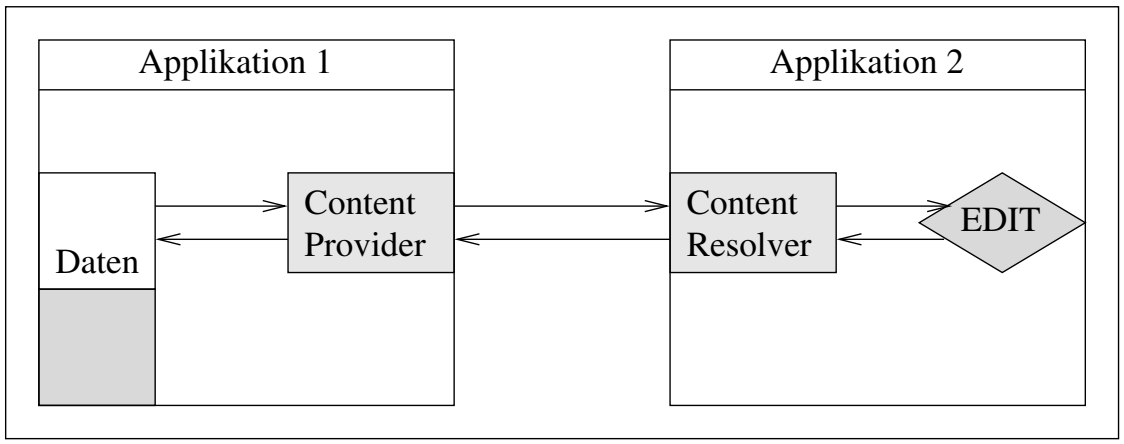

Abbildung 1: Funktionsweise Content Provider

3.3 Beispiel für bereits existierende Content Provider

Wie oben bereits beschrieben, liefert Android einige Content Provider schon mit. Sie ermöglichen den Zugriff auf einige Anwendungen, die zur Grundausstattung von Android gehört. Man kann diese im Paket android.provider finden. Diese sind zum Beispiel:

- Contacts: Mit Hilfe diese Providers können die Daten aus dem Adressbuch ausgelesen werden.
- MediaStore: Dieser Provider erlaubt den Zugriff auf die Multimedia Daten wie Musik, Video etc.
- CallLog: Wenn die Anruferliste von einem anderen Programm genutzt werden soll, kann man diesen Content Provider nutzen.
- Settings: Unter diesem Content Provider kann man auf die Systemeinstellungen zugreifen.

3.4 Universal Resource Identifier

Um auf einen Content Provider zuzugreifen, muss dessen Universal Resource Identifier (URI) in der Manifest.XML eingetragen sein. Dieser funktioniert dann ähnlich einer Webadresse als Ansprache eines bestimmten Content Providers. Wird er aufgerufen, liefert er einen Cursor auf die Daten zurück.

Der URI ist nach folgendem Schema aufgebaut:

```
Schema://Name/Beschreibung/ID
```

Das Schema gibt an, um welche Art der Datenquelle es sich handelt. Diese können sein:

- `android.resource`: Erlaubt den Zugriff auf das Environment.
- `file`: Gibt Zugriff auf eine Datei.
- `content`: Diese Art wird für den Content Provider benötigt. Der URI verweist auf einen Datenbank- oder Binärinhalt.

Der Name gibt die Klasse an, in der sich der Provider befindet. Hier seien als Beispiele genannt:

- `browser`: Erlaubt den Zugriff auf Daten des Browsers
- `calllog`: Erlaubt den Zugriff auf die Anruferliste

Die Beschreibung ist der Pfad, wo die Information zu finden ist. Diese kann zum Beispiel bei der Anruferliste `people` sein. Diese gibt die Namen von der Anruferliste zurück.

Die ID erlaubt den Zugriff auf einen bestimmten Zielwert. Zum Beispiel gibt `ID=2` den zweiten Datensatz zurück.

3.5 Ein Beispiel

Das folgende Beispiel stammt von der Android Development Seite[1] und zeigt, wie man gezielt auf einen Datenbestand zugreifen kann.

Gegeben ist folgende Datenbank:

[1] http://developer.android.com/guide/topics/providers/content-providers.html

ID	NUMBER	NUMKEY	LABEL	NAME	TYPE
13	(425) 555 6677	4255556677	Kirkland office	Bully Pulpit	`TYPE_WORK`
44	(212) 555-1234	2125551234	NY apartment	Alan Vain	`TYPE_HOME`
45	(212) 555-6657	2125556657	Downtown office	Alan Vain	`TYPE_MOBILE`
53	201.555.4433	2015554433	Love Nest	Rex Cars	`TYPE_HOME`

Nun möchten wir uns die Haupttelefonnummer der Personen anzeigen lassen. Das kann folgendermaßen aufgerufen werden:

```
import android.provider.Contacts.People;
import android.database.Cursor;

// Definiert ein Array für die Daten
String[] projection = new String[] {
                              People._ID,
                              People._COUNT,
                              People.NUMBER,
                              People.NAME
                            };

// Hier wird die URI auf dem PEOPLE Content Provider geholt
Uri contacts =  People.CONTENT_URI;

// Mithilfe des Cusors werden die Daten umgespeichert
Cursor managedCursor = managedQuery(contacts,//Die URI
                           projection, // Ziel für die Daten
                           null,       // Selektierte Zeilen
                           null,       // Selection arguments
                           // Sortiert die Namen in Aufsteigender
                           // Reihenfolge
                           People.NAME + " ASC");
```

Das Array `projection` sieht nun wie folgt aus:

ID	COUNT	NUMBER	NAME
13	3	(425) 555 6677	Bully Pulpit
44	3	(212) 555-1234	Alan Vain
45	3	(212) 555-6657	Alan Vain
53	3	201.555.4433	Rex Cars

Wenn nun eine Person hinzugefügt werden soll, müssen wir einen Content Provider benutzen:

```
import android.provider.Contacts.People;
import android.content.ContentResolver;
import android.content.ContentValues;

ContentValues values = new ContentValues();

// Add Abraham Lincoln to contacts and make him a favorite.
values.put(People.NAME, "Abraham Lincoln");
// 1 = the new contact is added to favorites
// 0 = the new contact is not added to favorites
values.put(People.STARRED, 1);

Uri uri = getContentResolver().insert(People.CONTENT_URI, values);
```

3.6 Sicherheit

Da Android es gestattet, eigene Software zu installieren, muss auch ein Sicherheitskonzept dahinterstehen, um das Ausführen von Schadcode zu erschweren.

Aus diesem Grund läuft jede Anwendung in einer eigenen Sandbox und kann keine anderen Programme unterbrechen oder stoppen. Zusätzlich muss jedes Programm von seinem Programmierer signiert werden.

Durch diese Signatur bekommt jedes Programm seine eigene User-ID zugewiesen, die es behält, solange es installiert ist. Die ID gibt den Pfad im internen Speicher an, wo die Dateien und die Datenbank der Anwendung gespeichert werden. Die entsprechenden Verzeichnisse werden als UNIX-Home-Directory behandelt, d. h. sie

lassen nur Zugriffe der entsprechenden User-ID zu. Dies schafft die Notwendigkeit für die Content Provider.

3.7 Vor- und Nachteile von Content Provider

Die Nutzung von Content Providern hat sowohl Vorteile, als auch Nachteile.

Ein Vorteil ist natürlich die eindeutige und allgemeine verbindliche Struktur von Schnittstellen. Dadurch ist die Interaktion von Anwendungen einfach zu gewährleisten. Des weiteren ermöglichen sie ein einheitliches Sicherheitskonzept.

Ein Nachteil wiederum kann sein, dass immer nur ein Cursor und nicht die Daten selber zurückgegeben werden. Ebenso müssen für jeden Content Provider alle Datenbankanfragen implementiert werden. Das ist gerade in solchen Fällen aufwendig, in denen es nur um einen einzigen Datensatz geht.

3.8 Alternativen

Es kann durchaus sein, dass man auf Content Provider verzichten möchte. Ein Grund dafür kann sein, dass man keinen Cursor geliefert bekommen möchte, sondernd direkt die Daten. Dafür kann man zum Beispiel einen Remote Service nutzen.

Remote Services sind Dienste, die im Hintergrund laufen, mit denen man über Binders kommunizieren kann. Diese Alternative kann man wählen, wenn man nur kleine Datenmengen von komplexer Datenstruktur übertragen will. Bei großen Datenmengen ist ein Content Provider wiederum performanter. Der Grund hierfür liegt darin, dass ein Remote Service ein eigeständiges Programm ist, das die ganze Zeit im Hintergrund laufen muss und somit Systemressourcen verbraucht.

Alternativ gibt es noch die Schnittstellen des `java.io` Paketes. Diese kann man aber nur nutzen, wenn man auf einen externen Datenspeicher zugreifen möchte. Auf die internen Daten oder gar die Datenbank eines anderen Programmes kann man damit nicht zugreifen.

4 Netzwerk-Schnittstellen

Ein Netzwerk auf einem mobilen Gerät zu benutzen bringt verschiedene Probleme mit sich.

Zum einen kann die Signalqualität stark variieren. Das gleiche gilt für die Datenrate. Schließlich kann ein Android-Gerät in einem Moment WLAN nutzen, im nächsten Moment aber UMTS. Verschiedene Netzwerkverbindungen können ebenfalls verschiedene Kostenstrukturen haben, was dazu führt, dass sie verschieden behandelt werden müssen. Android besitzt hierfür verschiedene Schnittstellen, die hilfreich sein können.

4.1 Netzwerk-Schnittstellen von Android

Android besitzt verschiedene Schnittstellenpakete.

4.1.1 Standard Java Schnittstellen

Android bietet zuerst einmal die Standardschnittstellen für Netzwerke des Java JDK (java.net). Über diese sind die Standardanwendungen der Netzwerkprogrammierung möglich. Genannt seien hier als Beispiele:

- ContentHandler: Diese Klasse erlaubt die Nutzung von MIME-Daten als Java Objekt.
- URL: Mit Hilfe dieser Klasse kann man auf eine Ressource im Internet zugreifen.
- SocketAdress: Durch diese Abstrakte Klasse kann man verschiede Protokolle implementieren.

4.1.2 Jakarta Commons HttpComponents

Die Bibliotheken von Jakarta Commons unterteilen sich in zwei Komponenten.

- Die Bibliothek Jakarta Commons HttpClient[1] (org.apache.http) bietet komfortable Unterstützung für vieles rund um das HTTP-Protokoll 1.0 und 1.1:
  - ↪ Alle HTTP-Methoden (GET, POST, PUT, DELETE, HEAD, OPTIONS, TRACE)

[1] http://jakarta.apache.org/commons/httpclient

  - ↪ Verschlüsselung mit HTTPS (HTTP über SSL)
  - ↪ Verbindungen durch HTTP-Proxys
  - ↪ Getunnelte HTTPS-Verbindungen durch HTTP-Proxys, via CONNECT
  - ↪ Verbindungen mit SOCKS-Proxys (Version 4 und 5)
  - ↪ Authentifizierung mit BASIC, Digest und NTLM (NT Lan Manager)
  - ↪ Multi-Part-Form POST

  Dies ist nur ein Teil der Möglichkeiten, die das Paket bietet.

- Die Bibliothek Jakarta Commons Net[2] implementiert bekannte Internet-Protokolle. Sie ist auf einem viel höheren Abstraktionsniveau als der HTTP-Client und unterstützt folgende Protokolle:
  - ↪ FTP (File Transfer Protocol). Dient der Dateiübertragung von und zu jedem beliebigen Rechner im Internet.
  - ↪ TFTP (Trivial File Transfer Protocol). Einfache Variante von FTP ohne Sicherheitsprüfung.
  - ↪ NNTP (Network News Transfer Protocol). Protokoll zum Versenden und Empfangen von Nachrichten in Diskussionsforen.
  - ↪ SMTP (Simple Mail Transfer Protocol). Standardprotokoll, mit dem E-Mails auf einen Server übertragen werden.
  - ↪ POP3 (Post Office Protocol, Version 3). Bisheriges Standardprotokoll, mit dem E-Mails vom Server abgeholt werden.
  - ↪ Telnet (Terminalemulation). Bietet die Möglichkeit, sich in spezielle Rechner einzuloggen.
  - ↪ Finger, Whois. Informations- und Nachschlagedienste, um Informationen über Personen einzuholen.

  Daneben unterstützt die Bibliothek auch die BSD-R-Kommandos wie rexec, rcmd/rshell und rlogin sowie Time (rdate) und Daytime. Für SMTP und POP3 ist die JavaMail API im Allgemeinen besser geeignet.

Eine detaillierte Aufzählung der Bibliotheken würde den Umfang dieser Arbeit übersteigen. Es sei hier auf die Seite von Apache verwiesen[3] oder auf das Buch „Java ist auch eine Insel“[4]

[2] http://jakarta.apache.org/commons/net/
[3] http://hc.apache.org/
[4] „Java ist auch eine Insel“, Galileo Computing, ISBN 978-3-8362-1146-8

4.1.3 Android spezifische Schnittstelle

Android liefert, zusätzlich zu den oben genannten Schnittstellen, eine Reihe von eigenen Schnittstellen mit. Zwei aus dem android.net Paket möchte ich hier besonders betrachten: den ConnectivityManager und die NetworkInfo. Außerdem möchte ich daran anschließend noch auf das android.net.wifi Paket eingehen.

Der ConnectivityManager ist ein Hintergrunddienst. Er bringt zwei hervorzuhebende Features mit sich. Zum einen sendet er im Falle eines Verbindungsabbruchs einen *Broadcast Intent* und informiert auf diese Art und Weise alle Programme, die über einen entsprechenden *Broadcast Receiver* verfügen, über den Verbindungsabbruch. Auf demselben Weg informiert er auch über die Wiederaufnahme von Verbindungen.

Als weiteres Feature bringt er ein Failover-System mit sich. Das bedeutet, er verwaltet alle Netzwerk-Verbindungen. Fällt eine aus (z. B.: Durch Verbindungsabbruch), versucht er automatisch auf einem anderen Weg eine neue aufzubauen.

Die andere Besonderheit ist die NetworkInfo. Die ist ein spezieller Datentyp, der durch einen ConnectivityManager erzeugt werden kann und Informationen über die Netzwerkverbindungen ausgibt. Dies kann sein, ob eine Verbindung hergestellt wurde, über welchen Netzwerk Adapter dies geschieht oder über welche Qualität das jeweilige Netzwerk verfügt.

Für WLAN liefert Android zusätzlich noch das Paket android.net.wifi mit. Dieses erlaubt einer Anwendung einen vereinfachten Umgang mit dem WLAN. Es ermöglicht zum Beispiel über die Klasse WifiManager.WifiLock einem Programm, eine WLAN-Verbindung aufrecht zu erhalten, so dass diese nicht durch den Energiesparmodus beendet wird. Auch kann mit Hilfe der Klasse ScanResult auf die Informationen von umliegenden Access Points zugegriffen werden. Insgesamt vereinfacht diese Klasse die gesamte WLAN-Abwicklung.

5 Fazit

Android liefert eine große Vielfalt an Schnittstellen mit. Es ergänzt die Standard-Java-Schnittstellen durch zusätzliche Pakete wie OpenGL im Grafikbereich oder den Apache-Commons im Netzwerkbereich und eigene Pakete wie android.net. Dies vereinfacht die Nutzung diverser Elemente wie WLAN oder standardisiert die Datenschnittstellen durch den Content Provider.

Dadurch wird eine schnellere und bessere Nutzung der Fähigkeiten eines Smartphones ermöglicht. Zum einen kommt hier der modulare Aufbau des Android Betriebssytems zum tragen, der durch die Content Provider noch unterstützt wird. Zum anderen ist es unnötig, komplette Implementationen von Strukturen, zum Beispiel im Bereich der Netzwerkverbindungen, vorzunehmen, da hier schon gute Implementationen vorhanden sind.

Die Dokumentation der Schnittstellen ist gut; viele Methoden sind selbsterklärend. Allerdings ist es nicht möglich, Anwendungen die für andere Mobiltelefone unter der Java ME geschrieben wurden, einfach auf ein Android-Gerät zu übertragen. In diesem Fall ist ein hoher manueller Anpassungsaufwand nötig. Es gibt zwar ein Programm, welches die Portierung übernehmen soll (J2ME Polish), dieses versagt aber, sobald irgendwelche speziellen Features benötigt werden, wie zum Beispiel Netzwerkverbindungen. Da es Byte-Code erzeugt, ist gezieltes Debuggen nicht möglich. Die einfachste und sauberste Lösung ist hier eine komplette Neuentwicklung des Programms.

6 Literaturverzeichnis

Die Informationen zum Vortrag, sowie zur Ausarbeitung:

- http://developer.android.com/index.html

  Die offizielle Android Homepage

- A. Becker M. Pant: Android, Grundlagen der Programmierung (1. Aufl.), dpunkt.verlag

  Ein gutes Buch, welches einen guten Einstieg in Android ermöglicht

- C. Ullenboom: Java ist auch eine Insel (8. Aufl.), Galileo Computing

  Dieses Buch habe ich für die Erläuterung zum Apache Commons HTTP Paket genutzt