Seminar:

Android: Plattform für mobile Geräte

Android Architektur

von Markku D. Lammerz

Wintersemester 2009/2010

Betreuer:
Julian Kunkel

Universität Hamburg
Department Informatik
MIN Fakultaet
AB Wissenschaftliches Rechnen

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Vorwort: Android, die Plattform

Android ist ein Open-Source-Betriebssystem für mobile Geräte wie Smartphones oder Netbooks, welches sogar im Rahmen von GoogleTV in Fernseher integriert werden [1] soll. Google trägt gemeinsam mit siebzig anderen Unternehmen im Rahmen der Open Handset Alliance die Entwicklung [2]. Android steht unter der Apache und GPL Lizenz und ist somit frei verwend- und änderbar. Die derzeit aktuelle Version ist 2.2 „Froyo“ (Stand Mai 2010), zu deren Neuerungen ein Dalvik JIT Compiler gehört, welcher einen Geschwindigkeitsgewinn um den Faktor zwei bis fünf mit sich bringt [3]. Ein nicht ausser Acht zu lassendes Problem ist, dass Geräte nicht immer auf die neuste Version geupdatet werden können. Bei manchen Geräten fehlt einfach die Speicherkapazität während bei anderen erst die herstellereigenen Benutzeroberflächen geupdatet werden müssen bevor ein Update möglich ist.

Die derzeit bekanntesten Smartphones, die Android benutzen, sind das Motorola Milestone (in den USA „Droid“ genannt), das Nexus One und das fast baugleiche HTC Desire. Seit dem Marktstart mit dem HTC Dream (in Deutschland T-Mobile G1) wächst der Marktanteil von Android rasant an. Der größte Verlierer auf dem Smartphone-Markt ist Windows Mobile, welches allein zwischen November 2009 und Februar 2010 vier Prozent ihres Marktanteils verlor, während Android über fünf Prozent dazugewann.

Top Smartphone Platforms 3Month Avg.			
	Share (%) of Smartphone Subscribers		
	Nov-09	Feb-10	Point Change
RIM	40,8%	42,1%	1,3
Apple	25,5%	25,4%	-0,1
Microsoft	19,1%	15,1%	-4,0
Google	2,8%	9,0%	5,2
Palm	7,2%	5,4%	-1,8

Quelle: comScore MobiLens

Mit den sogenannten Apps (Applications), die Apple mit ihrem iPhone SDK unterstützte und durch den Appstore populär machte, entwickelte sich ein neuer Markt für viele Entwickler, die sich bislang nicht an die Softwareentwicklung für Smartphones trauten. Android knüpft an diesen Erfolg mit seinem Android SDK an. So können Entwickler in Java Anwendungen programmieren und diese entweder selbst oder über den „Android Market“ veröffentlichen.

[1] GoogleTV bringt das netz ins Fernsehen, Heise News, `http://www.heise.de/newsticker/meldung/Google-TV-bringt-das-Netz-ins-Fernsehen-1004831.html`

[2] Open Handset Allaince FAQ, `http://www.openhandsetalliance.com/oha_faq.html`

[3] Android 2.2 Developers goodies, Android Developer, `http://android-developers.blogspot.com/2010/05/android-22-and-developers-goodies.html`

1.2 Die Entwicklungshilfen

Google stellt mit dem Android SDK alle Tools zur Verfügung, die benötigt werden um Anwendungen für die Android-Plattform zu entwickeln. Dazu gehören ein Debugger, ein Cross-Compiler, der Java-Bytecode in Dalvik-Bytecode umwandelt, und die Android Debug Bride, mit deren Hilfe direkt mit dem Gerät und dem Android Emulator interagiert werden kann. Wer gerne eine integrierte Entwicklungsumgebung (IDE) benutzt, kann Eclipse benutzen. Hierfür gibt es die Android Developer Tools (ADT) als Plugin. Die Anwendungsentwicklung findet in einer für Android ausgelegten Java Version statt, in der auf eine große Anzahl von Standard-Klassen zugegriffen werden kann und welche zusätzlich die Android-spezifischen Klassen beinhaltet, welche man z.B. für den Zugriff auf die Sensoren benötigt.

Java-Klassen	769
Android spezifisch	511
Andere	168

Google stellt unter `http://developer.android.com` eine sehr gute Dokumentation und verschiedene Tutorials für Einsteiger zur Verfügung.

Kapitel 2

Architektur

Die Architektur basiert auf einem Linuxbetriebsystem, in dem die Android Programme in den jeweiligen DalvikVM-Instanzen existieren und wirken. Die DalvikVM kann nun mithilfe des Kernels und der Libraries seinen Anwendungen alle Möglichkeiten der Hardware zur Verfügung stellen. Ausserhalb der VM dominiert C und C++, wobei die Anwendungsentwicklung in Java stattfindet. Im folgenden Abschnitt wird näher auf die einzelnen Schichten eingegangen.

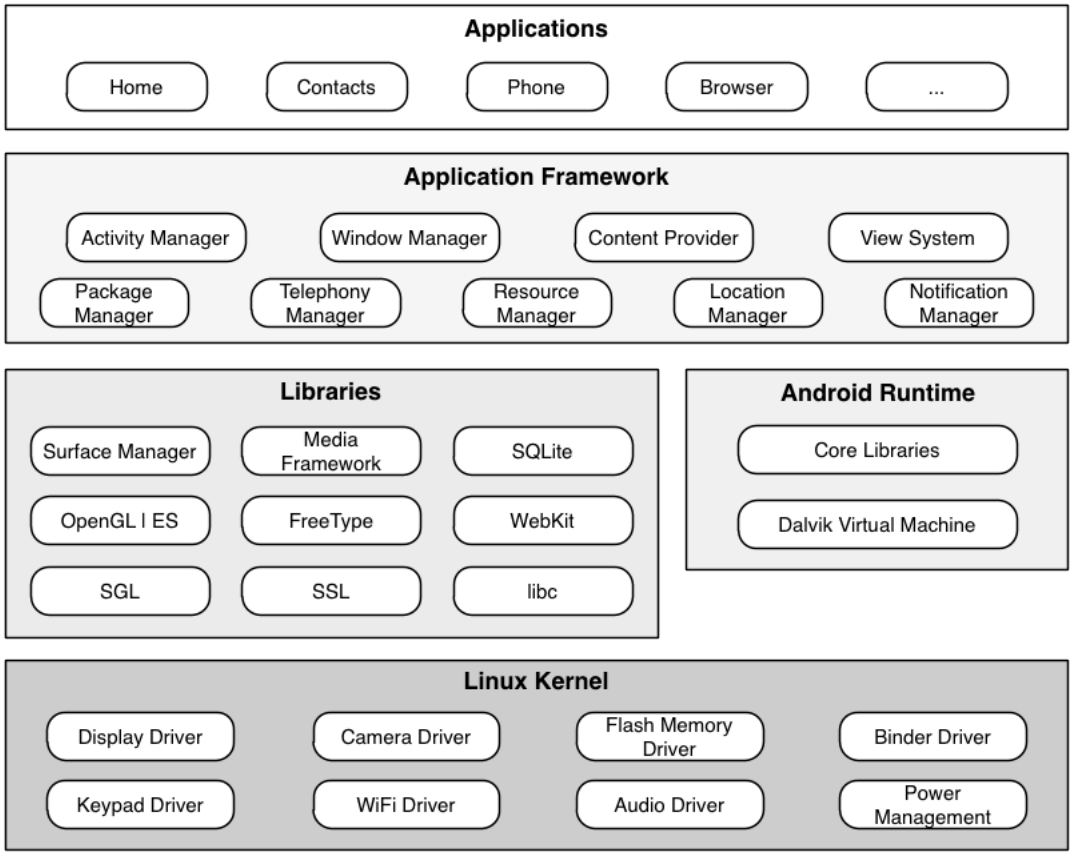

Abbildung 2.1: Android Architektur Überblick

2.1 Kernel

Der Linux Kernel (Version 2.6) dient zur Hardwareabstraktion. Zusätzlich beinhaltet sein Aufgabenbereich sich auf unterster Ebene um die Prozessverwaltung, Speicherverwaltung, Sicherheit, Netzwerk und das Energiemanagment zu kümmern.

2.2 Libraries

Die Libraries sind in C oder C++ geschrieben. Als fester Bestandteil des Betriebsystems bieten sie fast alles, was eine Android-Anwendung benötigen kann.

- Webseiten (WebKit):
- Datenbanken (SQLite)
  - unterstützt einen Großteil des SQL-92 Standards
  - komplett Public Domain (Rechteverzicht des Rechteinhabers)
- Video/Audio-Wiedergabe (Media Framework)
  - basiert auf „Open Core“ von PacketVideo
  - unterstützt: H.264,MP3,AAC,JPG,PNG,MPEG4 etc.
- 3D Anwendungen (OpenGL ES)
  - derzeitig: OpenGl ES 2.0
  - Hadwarebeschleunigung durch Grafikchip
- 2D Rendern (SGL)

2.3 Android Runtime

Die Hauptkomponente der Android Runtime ist die DalvikVM, eine Implementation einer JavaVM, basierend auf einer Registermaschine. Jede Android-Anwendung bekommt eine eigene Instanz dieser DalvikVM. Dies bietet zwei Vorteile; zum einen hat jede Anwendung einen eigenen Speicherbereich(Sicherheit), zum anderen kann das System einfach Anwendungen beenden(Skalierbarkeit). Da die ganze DalvikVM-Instanz beendet werden kann, entsteht somit das Problem mit blockierenden Anwendungen nicht. Zusätzlich bekommt jede Anwendung noch einen eigenen User. Somit sind die Daten einer Anwendung sowohl durch das Betriebssystem geschützt als auch durch die DalvikVM(Sandbox). Um Java-Code in einer DalvikVM laufen zu lassen, muss der Java-Bytecode mithilfe eines Cross-Compilers in Dalvik-Bytecode umgewandelt werden. Diese *.dex-Files sind unkomprimiert kleiner als *.jar-Files und können direkt in den Speicher gemapped werden. Damit wird CPU-Zeit gespart und somit Energie. Der DalvikVM wurde mit 2.2 der bis vor kurzem noch fehlende Just-in-time-Compiler hinzugefügt, welcher zu einer zwei-bis-fünf-fachen Geschwindkeitssteigerung führt.

2.4 Application Framework

Das Application Framework ist in Java geschrieben und stellt die Schnittstelle dar, mit der die Entwickler interagieren, wenn zum Beispiel die Anwendung auf die derzeitige Position zugreifen will(Location Manager). In den folgenden Abschnitten wird auf einige von diesen Frameworks eingegangen.

2.4.1 View System

Views sind die Hauptkomponenten des Userinterface und werden mithilfe von XML-Layout-Dateien manipuliert. Das View System ist zuständig für die Benutzerinteraktion, wie zum Beispiel die Erkennung von vordefinierten Gesten(Bsp.: Schüttelgeste).

2.4.2 Ressource Manager

Über den Ressourcenmanger ist der Zugriff auf Dateien, die im /res/-Ordner entweder als XML-Daten oder Binärdaten abgelegt wurden, mittels der R-Klasse möglich. Diese stellt auch spezielle Methoden zur Verfügung um die einzelnen Ressourcen, wie Bilder oder Texte, einzulesen. Im Abschnitt 3.3 wird nochmal genauer auf das Thema Ressourcen eingegangen.

2.4.3 Notification Manager

Durch Notifications kann der Benutzer auf neue Ereignisse aufmerksam gemacht werden, wie zum Beispiel eine neue E-Mail oder ein eingehender Anruf. Ein Icon kann in der Statusanzeige auftauchen, die LEDs leuchten oder der Vibrationsalarm des Geräts kann angesteuert werden. Es folgt ein kleines Beispiel:

```
(1) NotificationManger meinNotificationManger;
(2) meinNotificationManger=(NotificationManager)getSystemService(NOTIFICATION_SERVICE);
(3) Notifciation meiNotification=new Notification(icon, string, when);

(4) meinNotificationManager.notify(APP_ID, meineNotification);
```

In Zeile 1 wird eine Variable mit dem Namen meinNotificationManager angelegt, die den Typ `NotificationManger` hat. In Zeile 2 wird sich der `NOTIFICATION_SERVICE` vom System geholt und der eben erzeugten Variable zugewiesen. In Zeile 3 wird nun eine Notification angelegt, zur Veranschaulichung sind die Parameter nur Platzhalter. In Zeile 4 wird die Methode notify des `NotificationManagers` aufgerufen und die eben erzeugte `Notification` übergeben.

2.4.4 Location Manager

Über den Location Manager kann der Entwickler auf verschiedene Möglichkeiten zugreifen, um den derzeitigen Standort zu ermitteln. Ist die entsprechende Hardware verbaut, führt die GPS-Ortung zu den genauesten Ergebnissen. Ausserdem besteht die Möglichkeit, sich anhand der Funkmasten zu orientieren oder über die in der Umgebung befindlichen W-Lan-Netze. Die ersten beiden Optionen (GPS/Funkmast) sind derzeit in dem LocationManager leicht nutzbar.

```
(1) LocationManger locationManger;
(2) locationManger = (LocationManager) getSystemService(Context.LOCATION_SERVICE)
(3) Location location = locationManager.getCurrentLocation(„gps");

(4) String distanz = String.valueOf(location.diatanceTo(andereLocation)) + „ Meter";
```

In Zeile 1 wird die Variable `locationManager` definiert und in Zeile 2 wird sich wieder vom System ein SystemService geholt, diesmal der `LOCATION_SERVICE`. Nun wird in Zeile 3 in die Variable `location` vom Typ `Location` die per GPS ermittelte Position gespeichert. In Zeile 4 wird nun mittels der Hilfsmethode `distanceTo` von der Klasse Location, die derzeitige Position mit einer anderen Position verglichen und der Abstand in Metern als String zurückgegeben.

2.5 Applications

In der Architektur des Androidsystems sind die Applications (Anwendungen) austauschbar. Zwei Faktoren machen dies möglich. Erstens, dass die mitgelieferten Anwendungen nur auf die Schnittstellen (Application Framework) zugreifen, auf die auch „normale“ Entwickler Zugriff haben. Und zweitens die impliziten Intents welche im Abschnitt 3.1.3 behandelt werden.

Kapitel 3

Grundbegriffe

3.1 Komponententypen

Der folgende Abschnitt erläutert die einzelnen Android spezifische Komponenten aus denen sich eine Applikation zusammensetzt. Für weiterführende Informationen bietet sich die Android Developer Website an, auf der auch ausführlichere Beispiele zu finden sind.

3.1.1 Activities

Eine Activity repräsentiert einen Bildschirm. Wenn zwischen den Screens gewechselt werden soll, wird eine neue Activity aufgerufen und die vorherige pausiert. Eine Aktivity beaufsichtigt die Darstellung von Textelementen, Schaltflächen oder Menüpunkten.

3.1.2 Services

Services laufen ohne direkte Interaktion des Benutzers im Hintergrund. Mithilfe solcher Services kann Programmlogik ausgelagert werden. Entweder können sie in demselben Prozess mitlaufen (Local Service) oder in einem neuen Prozess, der selbständig arbeitet (Remote Service), erzeugen.

3.1.3 Intents

Bei Intents handelt es sich um Nachrichten auf Anwendungsebene, die z.B. eine Kommunikation zwischen Activities ermöglichen. Es gibt zwei Arten von Intents; die Expliziten und die Impliziten.

Explizite Intents

Bei einem expliziten Intent ist der Empfänger bekannt und wird direkt adressiert. Ein Beispiel dafür wäre:

```
(1) Intent meinIntent = new Intent(this, AufgerufeneActivity.class);
(2) meinIntent.setDate(foobar);
(3) meinIntent.putExtra(„feld1", „daten1");
(4) startActivity(meinIntent);
```

In Zeile 1 wird ein Intent erzeugt, dabei ist der erste Parameter der Context in dem sich die aufgerufene Klasse befindet(AufgerufeneActivity.class befindet sich im selben Paket) und der zweite der Name der aufgerufenen Activitie. Die Anweisungen in Zeile 2 und 3 dienen dazu extra Daten dem Intent mitzugeben, so können zum Beispiel Zwischenergebnisse zwischen Activities weitergegeben werden. In Zeile 4 wird nun die Activity mit Hilfe des Intents gestartet.

Implizite Intents

Bei impliziten Intents muss die Zielanwendung nicht bekannt sein. Das System entscheidet aufgrund der Parameter, welche Anwendung gestartet wird. Dadurch wird gewährleistet, dass die Anwendungen austauschbar gehalten werden.

```
(1) Intent meinIntent;
(2) meinIntent = new Intent(Intent.ACTION_VIEW, ContentUri.create(„http://wr.dkrz.de");
(3) startActivity(meinIntent);
```

In Zeile 1 wird die Variable `meinIntent` definiert und in Zeile 2 wird ein Intent erzeugt und zugewiesen. Als erster Parameter wird die Action `VIEW` übergeben. Es gibt eine Sammlung von Aktionen, die vordefiniert sind (siehe [4]) darunter `VIEW`, `DIAL` oder `WEB_SEARCH`. Das System wird nun anhand der Action und des zweiten Parameters erkennen(Intent Resolution), dass der Browser geöffnet werden soll.

3.1.4 Broadcast Receiver

Das System verschickt auch selbst Intents, wenn z.B. die Batterie fast leer ist. Jedoch sendet es nicht auf Anwendungsebene sondern auf Application-Framework-Ebene. Um diese zu empfangen benötigt man einen Broadcast Receiver.

```
(1) private class BatterieNotStatusReceiver extends BroadcastReceiver {
(2) @Override
(3) public void onReceive(Context context, Intent intent) {
(4) ... melde das Batterie fast leer ... }}
...
(5) IntentFilter mIF = new IntentFilter(„android.intent.action.BATTERY_LOW");
(6) BatterieNotStatusReceiver mBR = new BatterieNotStatusReceiver();
(7) context.registerReceiver(mBR, mIF);
```

In Zeile 1-4 wird ein eigener BroadcastReceiver definiert, dies geschieht durch das Erben von BroadcastReceiver. Um nun eigene Programmlogik ausführen zu lassen, wird die onReceive Methode überschrieben. Um diesen BroadcastReceiver beim System registrieren zu können, wird noch ein IntentFilter benötigt, durch den definiert wird auf welche Intents reagiert werden soll. Dies geschieht in Zeile 5, als Status auf den geachtet werden soll, wird ihm `BATTERY_LOW` übergeben. In Zeile 6 wird nun der eben definierte BroadcastReceiver erzeugt und in Zeile 7 dem Context mitgeteilt, dass er den BroadcastReceiver ansteuern soll wenn die im Filter hinterlegten Bedingungen erfüllt wrden.

3.1.5 Content Provider

Durch die Trennung jeder Anwendung mittels separater DalvikVM Instanzen sind auch die Speicherbereiche getrennt. Um nun z.B. an die Kontakte aus dem Adressbuch oder den nächsten Termin aus dem Kalender zu gelangen, wird eine Schnittstelle zwischen den Anwendungen benötigt. Diese stellt der Content Provider dar, welcher Daten vorhält um sie anderen Anwendungen zur Verfügung zu stellen. Ein Content Provider ist eindeutig identifizierbar und somit adressierbar durch seine URI.

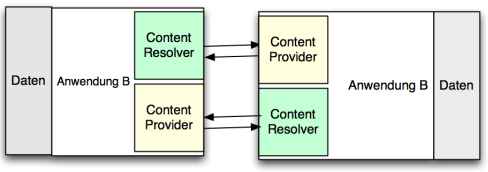

Abbildung 3.1: Content Provider

[4] Standard Activity Actions, ADev, `http://developer.android.com/reference/android/content/Intent.html`

3.2 Prozesshandhabung

Wie schon erwähnt hat jede Anwendung seine eigene DalvikVM Instanz und somit einen eigenen Linux-Prozess. In jedem dieser Prozesse können wieder mehrere Threads laufen, welche jeweils fest an ihren Prozess gekoppelt sind. Dabei gibt es eine Ausnahme. So können zwei Anwendungen, wenn sie von der selben Person signiert werden, in der selben VM laufen. Dies begünstigt zum Beispiel das Aufteilen einer Anwendung in einzelne Module.

Als Pluspunkt für Android gegenüber dem iPhone wird derzeit noch das Multitasking angeführt. Gerade wenn mehrere Anwendungen parallel laufen, muss verhindert werden, dass aufwendige Prozesse das gesamte System ausbremsen. Die Android-Strategie ist die Organisierung seiner Anwendungen oder besser seiner Activities in dem sogenannten „Activity Stack“:

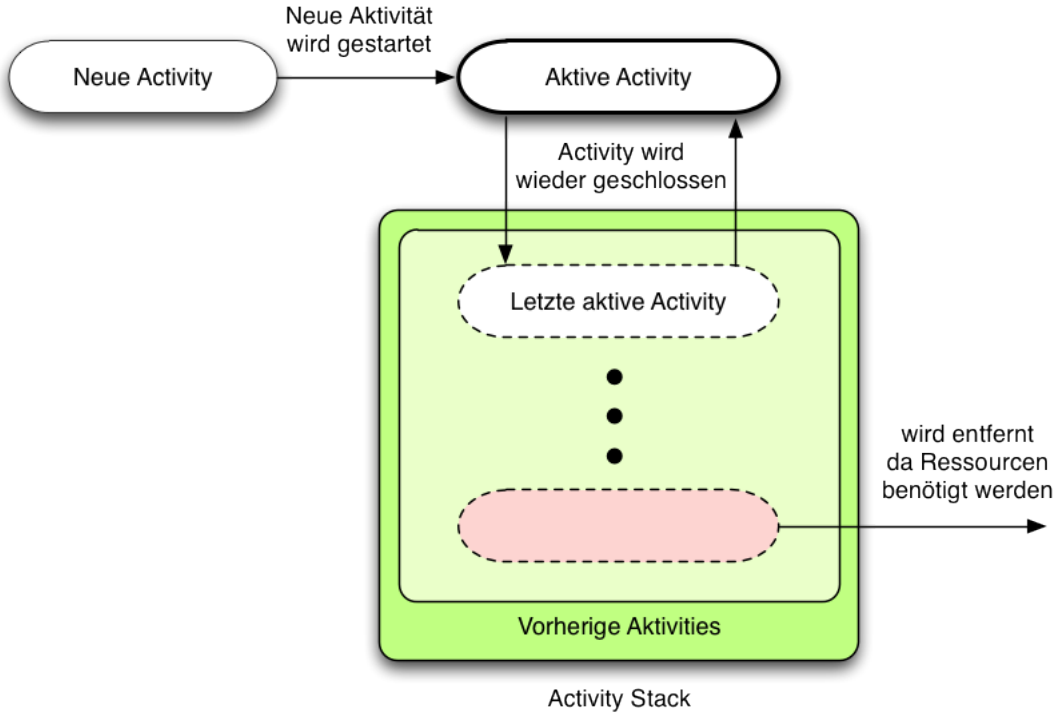

Abbildung 3.2: Activity Stack

In diesem Beispiel wird eine neue Activity gestartet, die derzeit aktive Activity landet auf dem Stack und rutscht bei jeder weiteren neuen Activity, die gestartet wird weiter nach unten. Wenn es nun zu Leistungsengpässen kommt, schließt Android die unterste Activity auf dem Stack.

Die folgende Grafik [5] beschreibt den Lebenszyklus einer Activity:

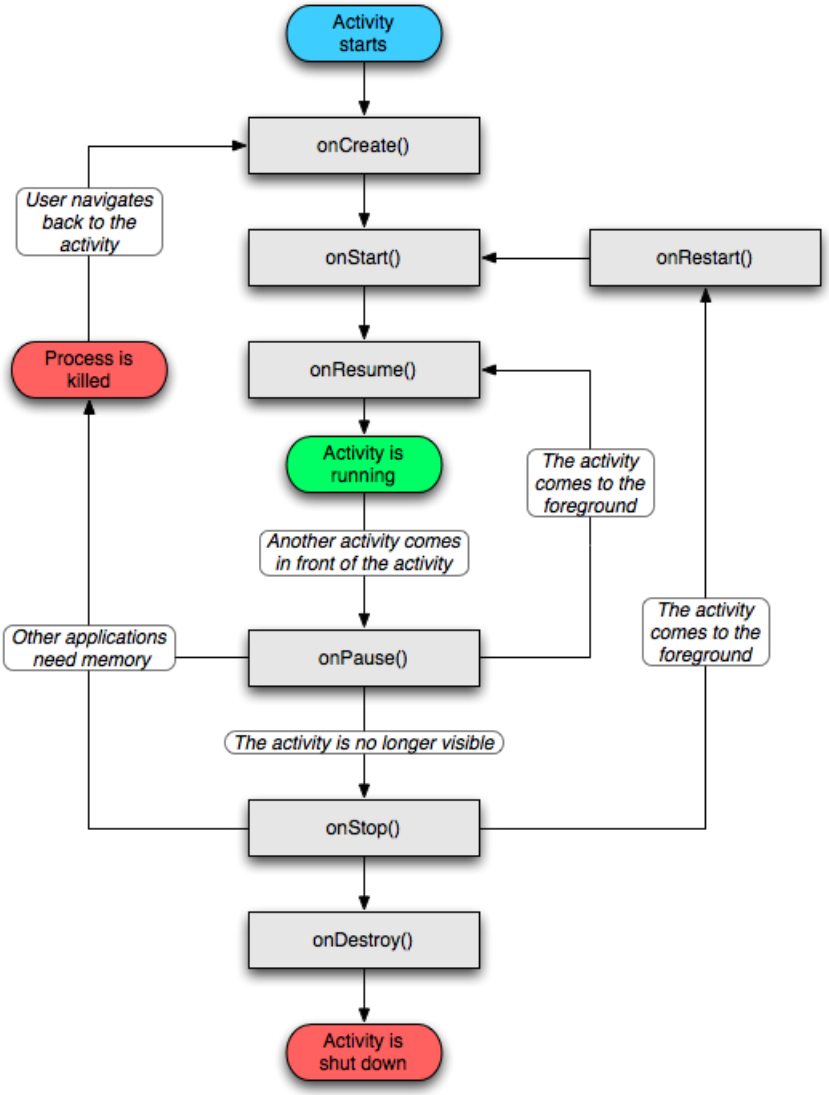

Abbildung 3.3: Activity Lebenszyklus

Eine Activity hat drei Zustände: `laufend`, `pausiert` und `gestoppt`. Sie ist pausiert, wenn sie nicht mehr `aktiv`, also nicht mehr angewählt, aber noch sichtbar ist. Zum Beispiel befindet sich eine Activity im Vordergrund, wird jedoch nicht im Vollbild-Modus dargestellt. Wurde die Activity `gestoppt`, ist sie nicht mehr sichtbar. Wenn nun das System Speicherplatz für eine andere Anwendung benötigt, wird der Prozess beendet. Navigiert der Anwender zu der geschlossenen Anwendung zurück, wird diese wieder gestartet. Dementsprechend wird eine `gestoppte` Activity restartet und eine `pausierte` fortgesetzt.

[5] Android Developer Guid, `http://www.developer.android.com/guide/`

Um zu verhindern, dass dem Anwender seine aktiven Anwendungen geschlossen werden, weil ein Service, der im Hintergrund läuft, meint, er bräuchte den gesamten Speicher, gibt es eine Priorisierung der einzelnen Prozesstypen.

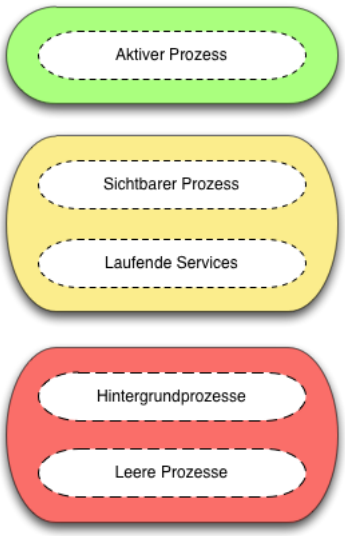

Abbildung 3.4: Prioritäten der einzelnen Prozesstypen

Ein aktiver Prozess hat die höchste Priorität, da der Benutzer nicht von sich neustartenden Prozessen gestört werden soll. Zu den sichtbaren Prozessen gehören `pausierte`. Danach kommen die `laufenden` Services und erst danach kommen Hintergrundprozesse, zu denen Broadcast Receiver und die `gestoppten` Activities zählen. Leere Prozesse sind sogenannte Phantomprozesse, die Android für sich bereithält, um Anwendungen schneller starten zu können.

3.3 Ressourcen

In jeder Anwendung werden entweder längere Texte, Bilder, Layouts oder Sounds verwendet. Diese ganzen Daten werden in Android durch die Ressourcen-Klasse (R-Klasse) verwaltet. Jedes Android Projekt hat einen /res/-Ordner, in dem z.B. Texte als XML-Dateien im Ordner /values/ gespeichert werden. Allgemein werden alle Daten durch XML-Dateien verwaltet[6]. Diese sind vom Menschen leicht lesbar und durch die Unterstützung von Editoren leicht zu erstellen.

In einem kleinen Beispiel wird nun demonstriert, wie ein Text gespeichert wird und innerhalb der Anwendung ausgelesen wird.

`/res/values/beliebigerName.xml`

```
<?xml version="1.0" encoding="utf-8"?>
  <resources>
    <string name="derText">"EINEN SCHÖNEN TEXT EINFALLEN LASSEN HIER"</string>
  </resources>
```

Der Dateiname spielt keine große Rolle. Wichtig ist der Name des Strings, über den auf das Element mit Hilfe der R-Klasse zugegriffen wird.

```
public final class R {
  public static final class string {
    public static final int derText=0x02040001;
  };
  public static final class layout {
    public static final int start_screen=0x02070000;
  };
  public static final class drawable {
    public static final int company_logo=0x02020005;
  };
};
```

Der Zugriff auf den Text gestaltet sich wie folgt:

```
String TextSpeicher = Resources.getText(R.string.derText);
```

Da die Daten beim Kompilieren in Binärdaten überführt werden, hat jeder Eintrag nun eine Adresse im Speicher. Durch die R-Klasse bleibt eine Typstruktur erhalten und die einzelnen Elemente sind durch ihren Namen auffindbar. Nun kann aus der Speicheradresse der Text ausgelesen werden. Die Resources-Klasse stellt ihre Hilfsfunktion zur Verfügung mit dem Namen `getText`. Ihr übergibt man die Adresse des Strings, welche in `R.string.derText` gespeichert ist.

[6] Reto Meier, Professional Android Application Development, wrox; 1 Aufl.; 2009; S. 56

Kapitel 4

Abschließende Worte

4.1 Zusammenfassung

Abschließend möchte ich die Kernthemen der Androidarchitektur noch einmal hervorheben. Android ist ein freies OS für Mobiltelefone, welches von Herstellern kostenfrei eingesetzt werden darf. Als Open-Source-Projekt gibt es viele Mitwirkende, die sich an der Weiterentwicklung beteiligen. Android besteht aus einem Linux-Kernel, der die Hardwareabstraktion durchführt und der DalvikVM, für die Entwickler in Java Anwendungen schreiben können. Da Java eine sehr verbreitete Programmiersprache ist - besonders in der Ausbildung an Universitäten - gibt es eine große Anzahl an Entwicklern die für Android entwickeln können. Die Java Version, die zur Anwendungsentwicklung genutzt wird ist der „normalen“ sehr nahe weist jedoch kleine Änderungen auf. So sind z.B. die Swing-GUI-Klassen nicht mehr enthalten aber dafür gibt es viele Androidspezifische Klassen, die sich z.B. auch um die GUI-Generierung kümmern.

Die Architektur von Android gestattet es, viele auch von der Hardware unterschiedliche Geräte mit dem selben Betriebsystem zu versorgen und dabei trotzdem die Möglichkeit zu bieten, dass Programme auf allen Geräten laufen. Hier ist auch das derzeit größte Problem. Durch die verschiedenen Versionen des Betriebsystems und den verschiedenen Hardwarespezifikationen müssen Entwickler einen großen Aufwand betreiben, um zu jedem Gerät kompatibel zu sein (in Bezug auf Auflösung und Leistungsfähigkeit des Geräts).

Für alle, die an der Entwicklung für Android interessiert sind, ist die Android-Dokumentation von Google auf `http://developer.android.com` sehr zu empfehlen, sie ist sehr ausführlich und einsteigerfreundlich. Weitere Literaturhinweise sind im Literaturverzeichnis zu finden.

Kapitel 5

Literaturverzeichnis

- **Android Developer Website von Google:** `http://developer.android.com`
  Die offizielle Dokumentation zu Android von Google die auch viele Beispiele bietet. Vergleichbar mit der Java-Dokumentation.

- **Android Application Development, Reto Meier, ISBN: 978-0-470-34471**
  Gutes englisches Buch das in die Androidprogrammierung einführt. Das Buch orientiert sich an einem Beispiel mitdem alle Teilbereiche erläutert werden.

- **Arno Becker Marcus Pant: Android, Grundlagen und Programmierung (1, Aufl.), dpunkt-verlag**
  Gutes deutsches Buch das in die Androidprogrammierung einführt. Das Buch orientiert sich an einem Beispiel mitdem alle Teilbereiche erläutert werden.