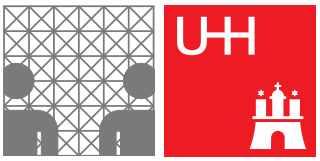
Programmiererfahrungen mit Android

Senad Ličina

Seminar Android: Plattform für mobile Geräte

Arbeitsbereich Wissenschaftliches Rechnen
Department Informatik
Uni Hamburg



Hamburg, 27.07.2010

Inhaltsverzeichnis

1 Einleitung

Im Wintersemester 2009/2010 wurde im Department Informatik der Universität Hamburg das Seminar "Android: Plattform für mobile Geräte" am Arbeitsbereich Wissenschaftliches Rechnen angeboten.
Im Zuge des Seminars haben sich alle Teilnehmer tiefgründig mit dem Betriebssystem Android beschäftigt. Das Seminar bestand aus einem Vortrag und einer Schriftlichen Ausarbeitung.
Das Interesse, an diesem Projekt teilzunehmen, war sowohl seitens der Studenten als auch seitens der wissenschaftlichen Mitarbeiter des Arbeitsbereichs Wissenschaftliches Rechnen sehr groß.

Bei diesem Dokument handelt es sich um die schriftliche Ausarbeitung von Senad Ličina zu dem Thema "Programmiererfahrungen mit Android".

1.1 Motivation

Als Informatik-Student und besitzer eines Android-Phones juckte es mich in den Fingern selber Software für Android zu schreiben. Dass dann sowohl ein Praktikum ("Mobile Computing" im Arbeitsbereich VSIS) als auch dieses Seminar, welche sich mit dem Thema auseinandersetzten, angeboten wurden war für mich natürlich ein Glückstreffer und stellte eine Gelegenheit dar mich näher mit dem Thema Android zu beschäftigen.
Durch die Teilnahme an diesem Seminar erhoffte ich mir einen detailierten Einblick in die Möglichkeiten die Android anbietet.

2 Projekte

Im folgenden werden die beiden Projekte vorgestellt, auf die ich mich in diesem Dokument beziehen werde. An beiden Projekten haben Senad Ličina und Arthur Thiessen gearbeitet.

2.1 Tetroid

Tetroid ist zu einem großen Teil in den letzten zwei Wochen des 3-wöchigen Blockpraktikums "Mobile Computing" (angeboten im Wintersemester 2009 vom Arbeitsbereich VSIS) entstanden. Es handelt sich dabei um einen Tetris-Klon.

2.2 HH-Plan

HH-Plan wurde für das Seminar "Android: Plattform für mobile Geräte" entwickelt. Es handelt sich dabei um eine Applikation, die -ähnlich wie geofox- Verbindungen mit den öffentlichen Verkehrsmitteln zwischen zwei Punkten zurückgeben kann.
Geplant war ein komplett selbstständiges Programm, welches seine Informationen von einem geofox-Server bezieht und diese anzeigt. Aus diesem Grund haben wir (Senad Ličina & Arthur Thiessen) beim "Hamburger Berater Team" (welches für geofox zuständig ist) um einen Zugang zu den geofox-Servern gebeten. Diesen hätten wir leider nur unter sehr strengen Bedingungen bekommen. Die uns angebotene Schnittstelle basiert auf SOAP. Da die Einarbeitung und Umsetzung mit der SOAP-Schnittstelle unseren zeitlichen Rahmen gesprengt hätte und dieser Zugriff uns nur zeitlich begrenzt gestattet werden würde, haben wir uns letztendlich gegen diese Form des Programms entschieden. HH-Plan füllt das Online-Formular aus, welches von geofox zur verfügung gestellt wird.
HH-Plan kann dabei die GPS-Position ermitteln und selbstständig eine Straße in der unmittelbaren Umgebung als Suchbegriff eintragen. Desweiteren ist ein Zugriff auf die Kontakt-Datenbank geplant.

3 Activities

Androidprogramme haben ein Ganz spiezielles Konzept wenn es um die Ausführung geht. Dieses Konzept wird mithilfe von Activities realisiert. Jedes Programm braucht mindestens eine Activity um gestartet zu werden.
Eine Activity ist eine "Programmeinheit" welche zum Ausführen von Aktionen benötigt wird. Ein Programm kann aus mehreren Activites bestehen, zeitgleich ist aber immer genau eine aktiv.
Activities werden verwendet, indem man eigene Klassen schreibt, welche die "Activity-Klasse" erweitern ("Listing 1", Zeile 1).
Es können mehrere Activities in den Hintergrund geschoben werden, wobei ihr aktueller Status zwischengespeichert und bei erneuter Aktivierung wiederhergestellt werden kann. Um dies zu realisieren stellt eine Activity immer Methoden bereit, die bei der jeweiligen Zustandsänderung aufgerufen werden. Diese Methoden werden dann in der eigenen Klasse überschrieben ("Listing 1", Zeilen 2-19).

```
1  public class MeineActivity extends Activity {
2    /** wird beim erstellen der Activity aufgerufen */
3    @Override
4    public void onCreate(Bundle savedInstanceState) {
5      super.onCreate(savedInstanceState);
6      setContentView(R.layout.hauptFenster);
7    }
8
9    /** wird beim pausieren der Activity aufgerufen */
10   @Override
11   public void onPause() {
12     * speicher den aktuellen Zustand *
13   }
14
15   /** wird beim reaktivieren der Activity aufgerufen */
16   @Override
17   public void onResume() {
18     * stelle den letzten Zustand wieder her *
19   }
20 }
```

Listing 1: Activity

Um das volle Potenzial von Activites auszunutzen sollte man eine eigene Activity für jedes "GUI-Fenster" erstellen.

3.1 Übergabe von Daten zwischen Activities

Eine Activity kann nicht direkt auf die Daten einer anderen zugreifen. Die Übergabe von Daten Zwischen Activities werden über ein sogenanntes Intent übergeben. Ein Intent ist die Beschreibung einer Operation, die ausgeführt werden soll. Intents werden zum Beispiel zum Wechsel zwischen Activities verwendet ("Listing 2", Zeile 1). Intents können "Extras" beinhalten. Extras werden über die Methode *putExtra(String name, Typ Objekt)* an einem Intent angefügt ("Listing 2", Zeile 2 & 3), wobei "putExtra" nur eine geringe Anzahl von Objekttypen (Beispielsweise String, int, long, double) annimmt.
In einer Activity kann dann über die Methode *getIntent()* auf das Intent mit dem die Activity aufgerufen wurde und mit der Methode *getExtras()* auf ein Bundle zugegriffen werden ("Listing 3", Zeile 1). In diesem Bundle befinden sich alle Daten die dem Intent vor dem Activity-Wechsel mit der Methode *putExtra()* hinzugefügt wurden.

```
1 Intent meinIntent = new Intent(this, NeueActivity.class);
2 meinIntent.putExtra("meinString", "have you tried to turn it
     OFF and ON again?");
3 meinIntent.putExtra("meinInteger", 42);
4 startActivity(meinIntent);
```

Listing 2: Activity welchsel: alte Activity

```
1 Bundle meinBundle = getIntent().getExtras();
2 String meinString = (String)bundle.get("meinString");
3 int meinInteger = (int)bundle.get("meinInteger");
```

Listing 3: Activity welchsel: neue Activity

In der Regel will man nicht nur Werte (wie Strings oder Zahlen) sondern Objekte von selbstgeschriebenen (oder anderen Komplexeren) Klassen übergeben. Um dies machen zu können müssen die zu übergebenden Objekte entweder Serializable bzw. Parcelable sein. Die zu übergebenden Objekte werden dann wie schon beschrieben über das "Extra-Bundle" im Intent übergeben.

3.1.1 Serializable

Bei *"java.io.Serializable"* handelt es sich um ein von Java vorgegebenes Interface. Es dient der "aufteilung" und "wiederherstellung" von Objekten. Wenn eine Klasse dieses Interface Implementiert, heißt dies dass Java Objekte dieser Klasse serialisieren darf.
Bei der Serialisierung werden Objekte die das Interface implementieren zu

einem Stream umgewandelt, beim Deserialisieren wird aus einem Stream ein Objekt gewonnen.

3.1.2 Parcelable

Auf Android-Telefonen kriegt man bei der Benutzung von *"java.io.Serializable"* ziemlich schnell ein großes Performance-Problem, weshalb man lieber zu einer Android optimierten Übergabe-Möglichkeit von Objekten greifen sollte.
Hier bietet Android uns das Interface *"android.os.Parcelable''* an. Parcelable ist ähnlich wie Serializable dafür zuständig, dass Objekte "aufgeteilt" und "wiederhergestellt" werden. Klassen, die Parcelable implementieren schreiben Ihre Exemplarvariablen in einen "Parcel" (im Grunde ein Container), aus diesem werden sie dann beim "wiederherstellen" gelesen. Um dies zu realisieren müssen bei Parcelable Methoden geschrieben werden.
Die Methode *writeToParcel(Parcel dest, int flags)* beschreibt, welche Daten in den Parcel geschrieben werden. die Methode *readFromParcel(Parcel in)* setzt das Objekt dann wieder zusammen, wobei die Daten in der gleichen Reihenfolge aus einem Parcel gelesen werden, wie sie auch reingekommen sind ("Listing 4").

```
1  @Override
2  public void writeToParcel(Parcel dest, int flags) {
3    dest.writeString(_meinString);
4    dest.writeInt(_meinInteger);
5  }
6
7  private void readFromParcel(Parcel in) {
8    _meinString = in.readString();
9    _meinInteger = in.readInt();
10 }
```

Listing 4: Parcelable

4 GUI Programmierung unter Android

Die Android-GUI besteht aus “View”- und “ViewGroup”-Objekten. Wobei ViewGroup-Objekte als Container für Views dienen und für die Anordnung dieser zuständig sind. Eine View repräsentiert jeweils eine einzelne Angezeigte UI-Komponente, ihr ist eine rechteckige Fläche zugewiesen und sie ist für das Zeichnen und die Event verarbeitung zuständig.

4.1 XML vs. Objekterzeugung

UI-Komponenten werden in Android entweder selber erzeugt und verwaltet oder man schreibt sich ein XML-Sheet, das dies für einen übernimmt.
Dabei wird XML empfohlen, weil es “leserlicher” ist und der Code durch die Strikte trennung der UI und des Programmiercodes an Übersichtlichkeit gewinnt.
Eine View (oder ViewGroup) wird durch jeweils einen XML-Tag repräsentiert, das Layout ergibt sich aus der Position des Tags, der ViewGroup und den Einstellungen, die im XML-Tag festgelegt werden. Aus dem Beispiel (“Listing 5”) wird ein UI erzeugt in der sich untereinander ein Textfeld und ein Button befinden. Die Option “android:id” wird dabei verwendet um aus dem laufenden Programm auf eine View zuzugreifen.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout android:layout_width="fill_parent"
3            android:layout_height="fill_parent"
4            android:orientation="vertical" >
5     <TextView android:id="@+id/text"
6            android:layout_width="wrap_content"
7            android:layout_height="wrap_content"
8            android:text="Ich bin ein Textfeld" />
9     <Button android:id="@+id/button"
10          android:layout_width="wrap_content"
11          android:layout_height="wrap_content"
12          android:text="Ich bin ein Button" />
13 </LinearLayout>
```

Listing 5: XML

4.2 GUI Objekte verändern

Wenn man die UI komplett über XML-Code generieren lässt, hat man das Problem dass man im Quelltext keine Referenz auf die UI-Objekte hat.

Ohne Referenz auf die Objekte kann man dementsprechend keine Methoden aufrufen und diese somit nicht mehr weiter verändern bzw. abfragen. Dieses "Problem" wurde gelöst indem eine weitere (automatisch generierte) Klasse (*"R"*) existiert, welche Verweise auf die einzelnen UI-Objekte hält.
Um diese Objekte zu erhalten, verwendet man die von "Activity" geerbte Methode *findViewById(int)* ("Listing 6", Zeile 1) welche ein View-Objekt zurückliefert. Um dieses zu benutzen ("Listing 6", Zeile 2) muss es vorher noch auf die spezifische Klasse gecasted werden.

```
1 _textView = (TextView)findViewById(R.id.text);
2 _textViewContent = _textView.getText();
```

Listing 6: findViewById

4.3 Event Verarbeitung

Die Event Verarbeitung wird über Listener realisiert. Diese müssen initialisiert und angemeldet ("Listing 7", Zeile 4) werden. Außerdem muss festgelegt werden, was bei welchem Event passieren soll. Dafür schreibt man die dafür vorgesehenen Methode um ("Listing 7", Zeile 7-9).

```
1  public class MeineActivity extends Activity implements
       OnClickListener {
2     protected void onCreate(Bundle savedValues) {
3        Button button = (Button)findViewById(R.id.meinButton);
4        button.setOnClickListener(this);
5     }
6
7     public void onClick(View v) {
8      // tue etwas
9     }
10 }
```

Listing 7: Event Verarbeitung

"Listing 7" zeigt ein Beispiel in dem die Activity selbst als Listener eingesetzt wird.

4.3.1 Threads

UI-Elemente können nur aus dem Haupt- bzw. UI-Thread heraus bearbeitet werden. Da man sich aber nicht immer im UI-Thread befindet, bietet jede

Activity die Methode *runOnUiThread(Runnable action)* an. Diese führt dann die Ihr gegebene Runnable im UI-Thread aus.

4.4 Grafikrealisierung bei Tetroid

Durch die von Tetris gegebene Spielgrafik bot sich vorerst ein *GridLayout* an, welches *ImageViews* beinhaltet. Da sich aber jedes Element eines *GridLayouts* selektieren ließ wurde diese Idee schnell verworfen. Statdessen habe ich mich für ein *FrameLayout* entschieden, auf dem *RectViews* "gezeichnet" werden.
RectView ist eine von mir geschriebene Klasse, welche ein *ShapeDrawable* in einer Quadratischen Form zeichnet. *ShapeDrawable* sind Views in einer bestimmten Form und Farbe.
Aus performancegründen werden alle Blöcke eines Steins auf ein eigenes *FrameLayout* "gezeichnet". Dadurch muss man das schon "feste" Spielfeld nicht jedes mal neu zeichnen sondern kann das spezifische *FrameLayout* mit dem aktuell "fallenden" Stein entfernen und (mit neuer Position) neu zeichnen.
Man kann sich Das Spielfeld in meiner Realisierung also in "Ebenen" vorstellen, wobei genau 3 Ebenen exisitieren. Die Ebene mit den schon festen Steinen, die Ebene mit dem aktuell fallenden Stein und die Ebene mit der "Vorschau" (welche anzeigt wo der Stein hinfällt, falls man ihn runterfallen lässt).

5 Services

Services sind Unterprogramme welche zu dem Prozess gehören, der sie gestartet hat. Services sollen dann eingesetzt werden, wenn eine Anwendung eine längerfristige Aktion ausführen oder Dienste anbieten soll. In der Regel laufen sie im Hintergrund und werden bei Bedarf abgefragt.
Android bietet von Haus aus eine vielzahl von Services an. Dabei handet es sich oft um Zugriff auf Sensordaten (wie zum Beispiel den Lagesensor oder den GPS-Sensor).
Man kann auch eigene Services schreiben, diese müssen von der Klasse "Service" erben.

5.1 Zugriff auf Services

Android benutzt ein ausgeklügeltes Rechtesystem. Vor der Installation werden die von der Applikation benutzten Rechte explizit aufgelistet. Dadurch ist eine große Transparenz über die verwendeten Mechanismen gewährleistet. Eine Applikation muss die gebrauchten Rechte vorher in ihrem Manifest festlegen ("Listing 8"). Das AndroidManifest ist eine XML-Datei, die jede Applikation besitzt. Eine Applikation wird durch ihr Manifest beschrieben. Es beinhaltet den Namen, die geforderten Zugriffsrechte und eine Auflistung der Activities & Services.

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

Listing 8: Event Verarbeitung

5.2 Verwertung von GPS-Koordinaten

Eine Anforderung an HH-Plan war, dass es die aktuelle Position als Start bzw. Ziel akzeptiert. Dies wurde mithilfe der GPS-Koordinaten realisiert. Da Geofox leider keine GPS-Koordinaten als Eingabe akzeptiert, wird vorher die am nächsten gelegene Adresse zu den aktuellen GPS-Koordinaten bestimmt. Dieses Verfahren bezeichnet man als "reverse geocoding".
Um die GPS-Koordinaten in dieser Art und weise zu verwenden, braucht man 4 zusätzliche Klassen.

- *"android.location.Location"*: repräsentiert eine geographische Position
- *"android.location.LocationManager"*: Der Systemservice, um auf die Position zuzugreifen

- *"android.location.LocationListener"*: Ein Listener, der bei veränderung der Position aktiviert wird. Wird beim *LocationManager* angemeldet.
- *"android.location.Geocoder"*: Kümmert sich um das geocoding bzw. reverse geocoding

Um im laufenden Porgramm Zugriff auf den LocationManager zu haben, muss man dies im Android Manifest festlegen und sich diesen mit der Methode *getSystemService(String)* holen ("Listing 9, Zeile 2"). Nun kann man sich die aktuelle (als letztes ermittelte) "Location" mit der Methode *getLastKnownLocation(String)* vom LocationManager geben lassen ("Listing 9, Zeile 3 & 4)".
Der LocationListener wird mit der Methode *requestLocationUpdates(String, int, int, LocationListener)* am LocationManager angemeldet ("Listing 9 , Zeile 5").

```
1 context = _currentActivity;
2 lm = (LocationManager)context.getSystemService(Context.
     LOCATION_SERVICE);
3 latitude = lm.getLastKnownLocation(LocationManager.
     GPS_PROVIDER).getLatitude();
4 longitude = lm.getLastKnownLocation(LocationManager.
     GPS_PROVIDER).getLongitude();
5 lm.requestLocationUpdates(LocationManager.GPS_PROVIDER
     ,5000,0,_myLocationListener);
```

Listing 9: Location initialisierung

Die Methode *onLocationChanged(Location)* des LocationListeners wird bei einer neuen Position aufgerufen ("Listing 10") . Bei der Anmeldung des LocationListeners können weitere Kriteren (wie z.B. minimale Entfernung zur letzten Location, minimale Zeit seit letztem Update) eingestellt werden.
Bei HH-Plan hat es genügt, dass der LocationListener den Längen- bzw. Breitengrad der neuen Location übernimmt.

```
1 public void onLocationChanged(Location location) {
2   latitude = location.getLatitude();
3   longitude = location.getLongitude();
4 }
```

Listing 10: LocationListener

Um aus der aktuellen Location die nächste Adresse herauszufinden braucht man zunächst ein Objekt der Klasse Geocoder ("Listing 11, Zeile 1"). Mit der Methode *getFromLocation(double, double, int)* kann man durch die Angabe

von Längen- & Breitengrad eine Liste von Adressen in der Nähe anfordern ("Listing 11, Zeile 2“).
Aus dieser Liste kann man nun eine Adresse wählen. Die Straße und Hausnummer erhält man durch ausführen der Methode *getAddressLine(0)* ("Listing 11, Zeile 3“).

```
1 gc = new Geocoder(_context);
2 addresslist = gc.getFromLocation(latitude, longitude, >
     maxResults<);
3 String addressNearToLocation = addresslist.get(0).
     getAddressLine(0);
```

Listing 11: reverse geocoding

6 Abschließendes

Android bietet Entwicklern eine einfache aber dennoch leistungsstarke Plattform. Das entwickeln und verbreiten von Software ist unkompliziert, weshalb sich ein tiefgründiger Blick in die Welt von Android lohnt!
Vielen dank an den Arbeitsbereich “Wissenschaftliches Rechnen”, der mir diesen Einblick in einem effizienem Seminar geboten hat.

7 Quellenangabe

- http://developer.android.com/
- http://shri.blog.kraya.co.uk/2010/04/26/android-parcel-data-to-pass-between-activities-using-parcelable-classes/