Google's Android: An Overview

Yoni Rabkin

yonirabkin@member.fsf.org

Abstract

This lecture is an overview of developing applications for Google's Android. We start by introducing Android and its components, we look at the anatomy of an Android application, we explain basic components of the Android application API including UI design and finally we say some things about the development environment.

Should I be giving this lecture?

I should because I...

- wrote one Android application from start to finish

I shouldn’t because I...

- wrote only one Android application from start to finish
- don’t know Java
- don’t use Eclipse

Legal Stuff

Portions of this work are reproduced from work created and shared by the Android Open Source Project and used according to terms described in the Creative Commons 2.5 Attribution License.

What is Android?

Android is a bunch of software[(a)] released by Google and the Open Handset Alliance[(b)].

[(a)] "platform" blah blah "framework" blah blah

[(b)] The alliance includes diverse members such as: China Mobile, Asus, T-Mobile (U.S.), Softbank Mobile (Japan), Motorola, Samsung, etc.

What isn’t Android?

- any specific piece of hardware
- “written entirely in Java”
- “runs only Java”

What does Android include?

- Linux Kernel (2.6) + drivers
- Android Runtime: core libraries + Dalvik virtual machine
- Libraries: OpenGL, SGL[a], Freetype, SSL, Sqlite, Webkit, libc
- XManager where X is some API component
- Pre-built and packaged applications: Contacts, Phone, Browser, Calendar, etc.[b]
- Android SDK

---

[a] 2D graphics

[b] http://source.android.com/

The Dalvik Virtual Machine

The Dalvik virtual machine...

- is a *register-based* Java VM
- is memory efficient
- is designed to run multiple VMs efficiently (one per application)[(a)]
- has no Just In Time compilation
- uses special byte-code
- relies on the Linux kernel for low-level stuff like threading

---

[(a)] communication via AIDL (Android Interface Definition Language)

Low-level Libraries

- BSD-derived libc tuned for small devices
- PacketVideo's OpenCore multimedia code[a] which supports: MPEG4, H.264, MP3, OGG, AAC, AMR, JPG, and PNG[b]
- libWebCore: Webkit based library to support the browser and web-views
- 3D support for hardware or software rendering via OpenGL
- Freetype font rendering
- Sqlite: relational database for application use

---

[a] http://www.packetvideo.com/products/core/index.html

[b] PNG is the standard for applications

App Anatomy: file structure

Selected files from a simple Android application structure.

```
./AndroidManifest.xml
./build.xml
./bin/Work.apk
./bin/classes/deliverator/foo/Work.class
./bin/classes/deliverator/foo/R.class
./res/layout/main.xml
./res/layout/setup.xml
./res/values/theme.xml
./res/values/strings.xml
./res/drawable/background.png
./res/drawable/overlay.png
./src/deliverator/foo/R.java
./src/deliverator/foo/Work.java
```


App Anatomy: file structure detail

Android can infer which resource to load by directory structure.

```
./res/values-en/strings.xml
./res/values-fr/strings.xml
```

..for languages or...

```
./res/drawable-en-rUS-finger/
./res/drawable-port/
./res/drawable-port-160dpi/
./res/drawable-qwerty/
```

...for layouts and graphics, which can lead to...

```
./res/drawable-en-rUS-port-160dpi-...-qwerty-dpad-480x320/
```


App Anatomy: AndroidManifest.xml

AndroidManifest.xml is an essential part of the application because it (amongst other things):

- names the Java package for the application
- describes activities, services, content providers, etc.
- declares permissions the application must have
- declares the minimum Android API level
- declares *Intents*

App Anatomy: Intents

Intents convey requests between all different components of the system.
The *Intent Resolution* mechanism revolves around matching an Intent against all of the <intent-filter> descriptions in the installed applications and *BroadcastReceiver*s.[a]

- defined in AndroidManifest.xml or dynamically
- belong to an *Activity* or *BroadcastReceiver*
- can deliver data via RFC2396 URIs

---

[a] multiple intents all get called!

Intents: Code Example

AndroidManifest.xml

```
<activity android:name=".Work" ... >
 <intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <action android:name="android.intent.action.SENDTO" />
  <category android:name="android.intent.category.DEFAULT" />
  <data android:scheme="smsto" /> ...
```

Work.java

```
Intent intent = getIntent();
String action = intent.getAction();
if (Intent.ACTION\_SENDTO.equals(action)) {
  doStuff(intent.getData().getEncodedSchemeSpecificPart() ...
```


Activities

An activity presents a visual user interface for a single task.

- always a subclass of the *Activity* base class
- one application, many activities
- one activity calls another (stack)[(a)]
- given a default window to draw in
- visual content defined by *Views*
- have a *life-cycle*

[(a)] An activity can return a value on exit, but doesn't have to.

Activities: Code Example

Suppose that your application has a message composition screen. This is an *Activity*. The UI would be implemented in its own file, say: *src/blah/blooey/Compose.java*:

```
public class Compose extends Activity
{
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    ...
```


Activities Life-Cycle: Code Example

Activities call one another, so our Compose activity would need to define what to do if it gets interrupted, calls another activity or gets called:

```
@Override
protected void onResume() {
   super.onResume();
   dosomething();}

@Override
protected void onPause() {
   super.onPause();
   dosomethingelse();}

protected void onDestroy() { ... }
protected void onStop() { ... }
```


Threads

Activity UI and your code run in the same process. So non-trivial computations require a separate thread.

```
final Handler mHandler = new Handler();
final Runnable mUpdateResults = new Runnable() {
  public void run() {
    updateResultsInUi();}};
...
Thread t = new Thread() {
  public void run() {
    mResults = doSomeThingHeavy(input);
    mHandler.post(mUpdateResults);}};
t.start();
...
private void updateResultsInUi() {
  if (mResults == Something) {updateUI(mResults);}}
```


Services

A service runs in the background for an indefinite period of time. A service is like an *Activity* without a UI.

- examples: getting data from the network, playing a video
- an *activity* can start a service
- starts explicitly with *Context.startService(intent)*
- starts implicitly with *Context.bindService(intent)*
- provides a callback via its *onBind()* method.
- runs until *Context.stopService()* or *stopSelf()*

Broadcast receivers

A broadcast receives and reacts to broadcast announcements.

- one application, many broadcast receivers
- always a subclass of *BroadcastReceiver* base class
- example broadcasts: low battery, call incoming, sms arrives etc.
- can start an *activity* or the *NotificationManager*
- as fickle and subtle as any asynchronous process is[a]

[a] example: registering and unregistering in onResume() and onPause()

Broadcast receivers: Code Example

in some service of the system:

```
Intent intent = new Intent(SOMEACTION);
intent.putExtra("name", "value");
sendBroadcast(intent);
```

...then in the app:

```
private BroadcastReceiver receiver = new BroadcastReceiver()
  public void onReceive(Context context, Intent intent) {
    String value = intent.getStringExtra("name");
    SomeActivityMethod(value);
```


Content providers

A content provider makes an application's data available to other applications.

- data can be from files, sqlite DB or anything else[(a)]
- always a subclass of *ContentProvider* base class
- use a *ContentResolver* object and call its methods

[(a)] *preferences* are lightweight name-value storage

Content providers: Code Example

```
String[] projection = new String[] {
    People._ID,
    People._COUNT,
    People.NAME,
    People.NUMBER};
Uri contacts =  People.CONTENT_URI;
Cursor managedCursor = managedQuery(contacts,
projection,
null,
null,
People.NAME + "_ASC");
```

managedCursor and *managedQuery* manage the cursor life-cycle for you.

UI: somelayout.xml

Stack some layouts with widgets in boxes:

```
<LinearLayout
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="wrap_content">
<TextView
android:id="@+id/sometext"
android:text="@string/howareyougentlemen"
android:layout_width="fill_parent"
android:layout_height="wrap_content" />
...
```


UI: SomeActivity.java

Call the layout and connect to the widgets:

```
public void setSomeText(String text) {
  TextView somewidget
  = (TextView) findViewById(R.id.sometext);
  somewidget.setText(text);
}
```


UI: gefingerpoken und mittengrabben

Find a button in the layout and connect it to a listener:

```
final Button somebutton
= (Button) findViewById(R.id.somebutton);
somebutton.setOnClickListener(new OnClickListener() {
  public void onClick(View v) {
    doSomething();
}});
```


The Development environment

Eclipse[(a)] is the natural environment for developing Android applications. However, *real* editors can be also used if you are a *real programmer*.

- The Android SDK integrates into Eclipse and helps the otherwise addled programmer write code.
- There are different versions of the SDK (1.0, 1.1, 1.5)[(b)]
- Three important command line tools: "android", "emulator" and "adb"

[(a)] I tried to load Eclipse for this lecture... it's still loading.

[(b)] at the time of writing

tools/android

An *avd* is an Android Virtual Device. Use *tools/android* to create, list, modify and remove *avd*s.

```
$ ./android create avd -t 3 -n "1.5lvl3" \
--sdcard /path/to/disk/image
$ ./android list
Name: 1.5lvl3
Path: /home/foo/.android/avd/1.5lvl3.avd
Target: Google APIs (Google Inc.)
Based on Android 1.5 (API level 3)
Skin: HVGA
Sdcard: /path/to/disk/image
```


tools/emulator and the Console

The emulator runs a virtual Android device.

```
$ ./emulator @1.5lvl3
```

Once the emulator is running, the Console (optionally) connects to an emulator instance and can issues commands or retrieve status.

```
$ telnet localhost 5554
Android Console: type 'help'
OK
```


Console Examples

Example Console commands:

- geo nmea GPGGA,123519,4807.038,N,01131.000,E ...
- redir add tcp:5000:5554
- power display/ac/status/present/health/capactiy
- network delay gprs/edge/umts/none
- network speed gsm/hscsd/gprs/.../full
- gsm call/accept/busy/.../list/voice/status
- sms send ...

Also from the command line:

```
$ ./emulator -netspeed gprs
```


tools/adb aka Android Debug Bridge

The Android Debug Bridge connects to a device or emulator.

```
$ ./adb -s HT93LLZ00513 shell
$ ./adb -s HT93LLZ00513 install /path/app.apk
```

... or connect to the device logs[a]

```
adb # logcat
I/DEBUG(551): debuggerd: Apr 21 2009
I/vold (550): Android Volume Daemon
...
```

... or connect to the device's sqlite DB

```
# sqlite3
SQLite version 3.5.9
sqlite>
```

[a] useful with Log.X(TAG, info)

Publishing

Publishing allows others to install your software. Publishing includes:

- sign the application (not with the debug key)[a]
- version the app: *android:versionCode="2"* and *android:versionName="1.1"*[b]
- provide *android:label="@string/app_name"* and *android:icon="@drawable/icon"*
- turn off debugging/logging, compile, sign and test
- upload to the Android market

---

[a] same signature: permissions and upgrades made easy

[b] accessible via: getPackageInfo(java.lang.String, int)

Getting the Source

The source (about 2GB) is hosted at *http://source.android.com/*, using Repo (some Python) and Git.

References and Adieu

- http://developer.android.com/
- http://android-developers.blogspot.com/
- http://groups.google.com/group/android-beginners
- http://groups.google.com/group/android-developers

Happy Hacking

