Deep Inside Android...

OpenExpo 2008 - Zurich
September 25th, 2008

Gilles Printemps - Senior Architect
Agenda

- What is Android?
- The Android platform
- Anatomy of an Android application
- Creating and deploying an application
What is Android?

A complete software stack for mobile devices.

Introducing Android
A first joined project of the Open Handset Alliance (OHA).
- First open, complete and free platform
- Software stack open-sourced under Apache 2.0 license
- Source code will be available and everyone will have the capability to build an image

The Android platform
It includes an operating system, a middleware and some applications.
- Lightweight and full featured
- Developers can extend and replace existing components

A generous development environment
A SDK is available to build, compile, test and debug user applications.
- Applications are developed using Java programming language
- No difference between the built-in applications and the user ones
Agenda

What is Android?

The Android platform

- Linux Kernel
- Native Libraries
- Android Runtime
- Application Framework...

Anatomy of an Android application

Creating and deploying an application
A Linux 2.6.24 fit for Android

Some common features have been removed
- No GBLIC support
- No native windowing system
- Does not include the full set of Linux utilities

New Android specific components have been added
- Alarm, “Android Shared Memory”
- “Kernel Memory Killer”, Kernel Debugger, Logger
**“Power Management”**

Based on the standard Linux Power Management, Android has its own component.
- Application uses user space library to inform the framework about its constrains.
- Constraints are implemented using lock mechanism.

**“Binder”**

Driver to facilitate inter-process communication between applications and services.
- A pool of threads is associated to each application to process incoming IPC
- The driver performs mapping of object between two processes
- “Binder” uses an object reference as an address in a process’s memory space
Android “Libc” implementation

A custom implementation, optimized for embedded use.
- BSD license
- Small size and fast code paths
- Very fast custom pthread implementation
- Built-in support for android-specific services (system properties, log capabilities)
- Doesn’t support some POSIX features
Native Libraries (cont.)

Storage, rendering, multimedia,…

Provides the main features on the Android platform:
- “SQLite”, a simple relational database management system (No IPC, single file,…)
- “WebKit”, an application framework that provides foundation for building a web browser
- “Media Framework”, based on PacketVideo openCORE platform (codec)
- Optimized 2D/3D graphic library based on OpenGL ES
- ...

“Surface Manager”

Provides a system-wide surface “composer” to render all the surfaces in a frame buffer.
- Can combined 2D and 3D surfaces
- Can use OpenGL ES and 2D hardware accelerator for its compositions
Native Libraries (cont.)

“Audio Manager”
Processes multiple audio streams into PCM audio out paths.
- Handle several types of devices (headphone, ear piece,…)
- Redirects audio stream to the specified output

“Hardware Abstraction Libraries”
Defines the interface that Android requires hardware “drivers” to implement.
- Set of standardized APIs the developer will have to implement
- Available for all the components a manufacturer can integrate on its Android platform
Dalvik Virtual Machine

An interpreter-only virtual machine (no JIT), register based.
- Optimized for low memory requirements
- Designed to allow multiple VM instances to run at one
- Relying on underlying OS for process isolation, memory management and threading support
- Executes Dalvik Executables (DEX) files which are zipped into an Android Package (APK)
Some essential services…

However, home made applications don’t interact directly with them.

- “Activity Manager” handles application lifecycle
- “Package Manager” holds information about applications loaded in the system
- “Windows Manager” handles all the application related windows
- “View system” provides all the standard widgets to build an application

- Hardware services provide access to lower-level hardware APIs: Bluetooth, telephony, location,…
Platform initialization

The bootloader loads the kernel and starts the init process

Daemons for handling low level hardware interfaces are started up (usb, adb, debugger, radio)

“Zygote”, the initial Dalvik VM process is created

“Runtime” process initiates the “Service Manager”, a key element for “Binders” and IPC communication

“Runtime” process requests “Zygote” to start a new instance of Dalvik for running the “System Server”

The two first processes are able to handle graphic and audio outputs

All the others android components are then started
Agenda

What is Android?
The Android platform

Anatomy of an Android application

Activity
Service
Content Provider
Processes and Tasks...

Creating and deploying an application
One single screen in an Android application

What is really an “Activity”?

Displays a user interface component and responds to system/user initiated.
- When an application has a user interface, it contains one or more “Activities”
  One activity is then considered as the main entry point by the system
- An existing “Activity” can be replaced with a new one that fulfill the same contract
- Each “Activity” can be invoked from others applications

Adding a new “Activity” in an Android project
- The new Java class must extend the framework “Activity” class
- Created “Activity” must be defined into the application’s Manifest

```
<activity android:name="sampleActivity"
  android:label="@string/app_name">
  ...
</activity>
```
“Activity” Lifecycle

Android maintains an history stack of all the activities which are spawned in an application

A “Classic” scenario

An “Activity” is started and explicitly finalized (i.e. finish() is called).

Start up

- OnCreate():
  First method called during lifetime; the “Activity” is created
  ➔ Initialize all the resources (i.e. create a thread, connection to a database)

- OnStart():
  Execution of the “Activity” can start anytime; system put it on top of the stack
  ➔ Activity is on-screen (may be not in foreground); user cannot interact with.

- OnResume():
  Activity is visible to the user and has the focus.

Shutdown

- OnPause():
  Focus is lost and potentially the “Activity” will shutdown
  ➔ Save data (persistence)

- OnStop():
  “Activity” is no more on the top of the stack

- OnDestroy():
  “Activity does not exist anymore in the system
  ➔ Resources can be freed
“Activity” Lifecycle (cont.)

Switching between “Activities”
Android will notify the running “Activity” through two new callbacks.

- OnFreeze(): All the “Activity” to save UI state
- OnStart(): Save the value belonging to the UI (i.e. field in a form)
- OnRestart(): Signal the “Activity” will be restarted

Dropping an “Activity”
An “Activity” can be dropped from memory only in the following specific states:
- When it is paused, even if it is completely alive
- When it is completely obscured by another “Activity” (stopped)

The system can:
- Ask the user to finish it
- Directly kill the process it belongs to
**Navigation and Triggering**

**“Intents / Intent Filters”**

Simple message objects that represent:

- An intention to do something
- A declaration of capacity and interest in offering assistance to those in need

An “intent” is made up a number of pieces of information describing the action or the service:

- “action” attribute is typically a verb (*VIEW, EDIT, DIAL, …*)
- The data to operate on is expressed in the form of an Universal Resource Identifier (URI)
- “category” attribute gives additional information about the action to execute

```xml
<intent-filter>
    <action android:value="android.intent.action.VIEW" />
    <data android:mimeType="com.esmertec.player/media/12" />
</intent-filter>
```
“Broadcast Intent Receiver”

Broadcasted notifications from the system

When using these “Receivers”?

When an application desires to receive and respond to a global event.
- In order to be triggered when an event occurs, application does not have to be running
- By default, Android includes some built-in “Intents Receiver”

How to receive these specific “Intents”?
- The user class must extend the framework “IntentReceiver” class
- To process incoming “Intents”, “onReceiveIntent()” method is implemented
- “receiver” element must be described into the application’s Manifest

```
<receiver android:name=".Alerter">
  <intent-filter>
    <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
  </intent-filter>
</receiver>
```
A long-running background task

Defining and invoking a “Service”

Adding a “Service” with Android is quite similar than for an “Activity”.
- Framework “Service” class must be extended
- The new “Service” must be defined into the application’s Manifest

```
<service class="AdderServiceImpl"/>
```

Class name

When a “Service” is using IPC, an AIDL description of its features is also needed.
- Android Interface Definition Language (AIDL) is used to generated code to allow communication between two processes through IPC
- This mechanism is interface-based, similar to Corba (Unix) or COM (windows) but lighter weight
- With this specific mode, the “Service” is started through the binder and not with “startService()”
Handle data and expose them to other applications

Why using a “Content Provider”?

The only way to share data between Android applications.
- It implements a standard set of methods to allow access to a data store
- Any form of data storage can be used like SQLite database, files, or memory hash map

A “Content Providers” exposes a unique URI used to query, add, update and delete data:
- A standard prefix (“content://”)
- The authority part (fully-qualified to ensure uniqueness)
- The path to determine what kind of data is being requested
- A specific record being requested, if any
A relation between the kernel and Android

Processes
A low-level kernel process in which an application’s code is running.
- By default, Android binds the content of an APK to a Linux process
- “Process” tag can be used to tune this relation with a lower granularity (activity, service, …)

Tasks
A notion that users know on other platform as “application”.
- A collection of related “Activities”
- Capable of spanning multiple processes
- Interaction with “Activities” can be controlled through “Activity’s launchMode” attribute
Agenda

What is Android?

The Android platform

Anatomy of an Android application

Creating and deploying an application

- Eclipse plug-in
- Building the package
- Running the application
- Debugging environment...
Creating an Application

Using the “Eclipse” plug-in

After some clicks, a “basic” application can be created, compiled and executed.

- Only some properties are required to create an Android skeleton
- The plug-in creates all the file/directories useful to generate the final APK
- It can be freely downloaded by following the instructions at

http://code.google.com/android/intro/installing.html

An element of a complete development environment containing:

- An Android emulator which mimics the hardware and the software of a mobile phone
- Some tools for debugging, tracing, logging…
Creating an Application (cont.)

“AndroidManifest.xml” file

Describes the application components that the package exposes.

- This file is mandatory and is located at the root of the application directory
- A top-level application presented to the user must include at least one “Activity” that supports the “MAIN” action and “LAUNCHER” category

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.esmertec.openexpo"
    android:versionCode="1" android:versionName="1.0.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".SampleApp" android:label="@string/act_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="act_name">SampleActivity</string>
    <string name="app_name">Android Application</string>
</resources>
Building an APK...

**Dalvik Executable (DEX files)**

A optimized format for efficient storage and memory mappable definition.
- All classes are compiled with a standard Java language compiler
- Migration between a standard class file and the Android bytecode is done with the “dx” tool
  (For debugging purpose, it can also present a people-friendly readable format)

```
* AIDL  |  * Java
XML +  res +  JAR +  Java = Application
Manifest + Android + R
```

Generating the Android specific java files:
- When an application contains IPC services, AIDL files must be preprocessed to get Java classes
- Interface based AIDL tool saves developer the time used to write marshalling code

- All Android projects have a resource ID file called “R.java”
- It’s basically a series of integers that can be used to reference the actual resources
- This file is generated automatically by “aapt” and should not be edited
Building an APK... (cont.)

Packaging the resources

During the build process, an archive containing application resources is also created.
- This file is automatically generated by the “aapt” utility
- Among other things, it’s made up of an index resource file called “resources.arsc”

Generating the Android PacKage (APK)

Typically, an APK includes all the files related to a single Android application:
- A compressed collection of “AndroidManifest.xml”, application code, resources...
- Additionally, “META_INF” directory can integrate certificates used to sign application
Installing / Removing a package

All Android applications are handled by the “Package Manager”.

- An APK must just be uploaded to a specific directory
- Two locations are monitored: “/data/app” (user) and “/system/app” (system)

Android Debug Bridge (ADB)

A client/server application used as a gateway to communicate with the device. It allows:

- Copying files to/from the device or the emulator (“push”, “pull”, “install”)
- Running shell binary commands
Debugging environment

**Dalvik Debug Monitor Service (DDMS)**
A GUI application providing debugging information to Android developers:
- Processes and threads examination
- Memory heap statistics and allocation tracker
- Emulator controls to simulate specific device events
- File explorer to perform basic management on the file system
- Dump of the printed out messages,...