Journal of Telematics and Informatics (JTI)
Vol.1, No.2, September 2013, pp. 89~96
ISSN: 2303-3703

A Study on Android Application Development

Rajkumar A. Soni
MCA Dept., Laljibhai Chaturbhai Institute of Technology, Bhandu, India
email: rajkumar.soni@lcit.org

Abstract

As and when the world moves towards the hand-held devices, there should be an innovation as well as awareness of such kind of application development used for the hand-held device. As time elapses the world started moves on the fingers using such kind of developed applications. In this paper we discuss the most fundamental attributes and crucial steps followed to create a robust application using Android OS. This kind of contribution will help a lot to the initiators who want to switch-over from other technologies to the mobile application development.

Keywords*: Dalvik Virtual Machine, Open Source, Vendor Neutral, User Interface, Kernel*

1. Introduction

Along with the great demand of mobile-applications in the era of pervasive and ubiquitous computing, a number of issues related to [1] qualities of mobile-applications have also gained a great attention in mobile-application development process. Great competitions in the mobile-application business compel mobile-application developers more quality conscious. It will be of great value if a mobile-application could survive against demanding and changing customer requirements, and changing business requirements.

Building high quality mobile-application is really a difficult and challenging task. But the support for right development process, methods, tools, and people really make it possible to achieve high quality of mobile-application. As the development platform influences associated development process, methods, tools, and people, it really plays a major role for making development process simple, efficient, and robust and for achieving high quality of mobile-application. Android platform [2] is open and standard based platform on which a scalable mobile applications can be developed. As the applications targeted on Android platform are vendor neutral, the organization does not face the problem of vendor lock-in.

Android SDK comprises of several features like open platform, WI-FI hardware access, Full communication stack (GSM, EDGE, 3G, Bluetooth), GPS, Multimedia, IPC messaging, share data stores, web-kit browser, P2P via google talk, Eventually hardware acceleration 3D graphics

OPEN GL ES is Media Libraries, and open application framework (reuse and replacement). The presented work, in this paper, concentrates on achieving high quality for mobile-application developed on Android platform from different perspective of overall development process. The related work in the category is as follows.

The Section II discusses about Android Architecture. The Section III represents how to install and configure Android SDK with Eclipse IDE. The Section IV presents whole Anatomy of an Android Application. Section V presents Android User Interfaces. Section VI presents AndroidManifest.xml file. Section VII presents Resources and finally the Section VI presents the conclusions.

2. Android Architecture

Android is an open source mobile device operating system developed by Google based on the Linux 2.6 kernel. The Linux kernel was chosen due to its proven driver model, existing drivers, memory and process management, networking support along with other core operating system services [3]. In addition to the Linux kernel various libraries were added to the platform in order to support higher functionality. Many of these libraries originate from open source projects; however the Android team created their own C library, for example, Bionic C-Library which contains Unique Components designed for Android and consume Only 200k (half the size

Received May 29, 2013; Revised August 2, 2013; Accepted August 26, 2013

of std libc). They also developed their own Java runtime engine, optimized for the limited resources available on a mobile platform called the "Dalvik Virtual Machine." Lastly, the application framework was created in order to provide the system libraries in a concise manner to the end-user applications. It also provides higher-level Services to Applications using Java Classes. Applications contains Standard Applications come with System, SDK to develop new applications, C/C++ Applications can be run as well, Eclipse based Application Development Environment is also to be provided (with special plug-in). In this survey we highlight the major areas which should be understood precisely to develop an Android application development.

Figure 1. Android Architecture [1]

Linux kernel comprises of Hardware abstraction layer which is responsible to provide security, Memory management, Process management, Network stack, Driver model. We can not make linux calls directly but we can use some utilities like adb shell to examine file system, or to view active processes. Libraries having support for surface management, 2D and 3D graphics, Media Codecs, SQLite Database and browser engine.

Dalvik Virtual machine is a google's implementation of java which is specifically optimized for mobile devices. It runs .dex files instead of .class file which is more compact and efficient than the later one. Concerns about memory and power consumption are also to be taken care.

Application Framework are the crucial blocks for creating an android application, its pre-installed in android os. Activity Manager controls application life cycle, "backstack". Content Providers encapsulates any data to share between applications, Resource manager providing access to non-code resources such as localized strings, graphics, and layout files. Location manager figures out the location of the phone (GPS, GSM, Wi-Fi). Notification Manager enables all applications to display custom alerts in the status bar.

Applications are the programs which take over the entire screen. An application can be alive even if its process has been killed.

3. Install and Configure Android sdk with Eclipse Ide

Android uses the java programming language and for anyone who has learned Java, to develop the basic android application is quite easy. First you will need to download [8] JDK ,then you have to download the [4] Android SDK and if you are using [5] Eclipse (IDE) then install the [6] ADT Plug-in, after installing the ADT Plug-in go to Eclipse Menu's Help-> Software Updates-> Available Software-> Add Site. In the Location Field you can provide either [7] URL or can choose the ADT from local machine. After that you have to select all "Developer Tools" provided under the site you added, then click on Install button. Switch to Menu's Windows-> Preferences->Android, click on browse button and go to the location where android SDK is installed on your machine. You can see platforms installed after hitting the "apply" button. Now your Eclipse IDE is ready to develop an Android application.

4. Anatomy of an Android Application

There are four building blocks to an Android application:

1) Activity
2) Intent
3) Service
4) Content Provider

Not every application needs to have all four, but your application will be written with some combination of these. Once you have decided what components you need for your application, you should list them in a file called AndroidManifest.xml. This is an XML file where you declare the components of your application and what their capabilities and requirements are.

1). Activity

Activities are the most common of the four Android building blocks. An activity is usually a single screen in your application. Each activity is implemented as a single class that extends the Activity base class. Your class will display a user interface composed of Views and respond to events. Most applications consist of multiple screens. For example, a text messaging application might have one screen that shows a list of contacts to send messages to, a second screen to write the message to the chosen contact, and other screens to review old messages or change settings. Each of these screens would be implemented as an activity. Moving to another screen is accomplished by a starting a new activity. In some cases an Activity may return a value to the previous activity - for example an activity that lets the user pick a photo would return the chosen photo to the caller. When a new screen opens, the previous screen is paused and put onto a history stack. The user can navigate backward through previously opened screens in the history. Screens can also choose to be removed from the history stack when it would be inappropriate for them to remain. Android retains history stacks for each application launched from the home screen.

2). Intent and Intent Filters

Android uses a special class called Intent to move from screen to screen. Intent describes what an application wants to do. The two most important parts of the intent data structure are the action and the data to act upon. Typical values for action are MAIN (the front door of the application), VIEW, PICK, EDIT, etc. The data is expressed as a Uniform Resource Indicator (URI). For example, to view a website in the browser, you would create an Intent with the VIEW action and the data set to a Website-URI.

```
new Intent(android.content.Intent.VIEW_ACTION, ContentURI.create("http://developer.android.com"))
```

There is a related class called an IntentFilter. While an intent is effectively a request to do something, an intent filter is a description of what intents an activity is capable of handling. An activity that is able to display contact information for a person would publish an IntentFilter that said that it knows how to handle the action VIEW when applied to data representing a person. Activities publish their IntentFilters in the AndroidManifest.xml file. Navigating from screen to screen is accomplished by resolving intents. To navigate forward, an activity calls startActivity(myIntent). The system then looks at the intent filters for all installed applications and picks the activity whose intent filters best matches myIntent. The new activity is informed of the

intent, which causes it to be launched. The process of resolving intents happens at run time when startActivity is called, which offers two key benefits:

- Activities can reuse functionality from other components simply by making a request in the form of an Intent
- Activities can be replaced at any time by a new Activity with an equivalent IntentFilter

Intent Receiver

You can use an IntentReceiver when you want code in your application to execute in reaction to an external event, for example, when the phone rings, or when the data network is available, or when it's midnight. Intent receivers do not display a UI, although they may display Notifications to alert the user if something interesting has happened. Intent receivers are also registered in AndroidManifest.xml, but you can also register them from code using Context.registerReceiver(). Your application does not have to be running for its intent receivers to be called; the system will start your application, if necessary, when an intent receiver is triggered. Applications can also send their own intent broadcasts to others with Context.broadcastIntent().

- **Activity LifeCycle**

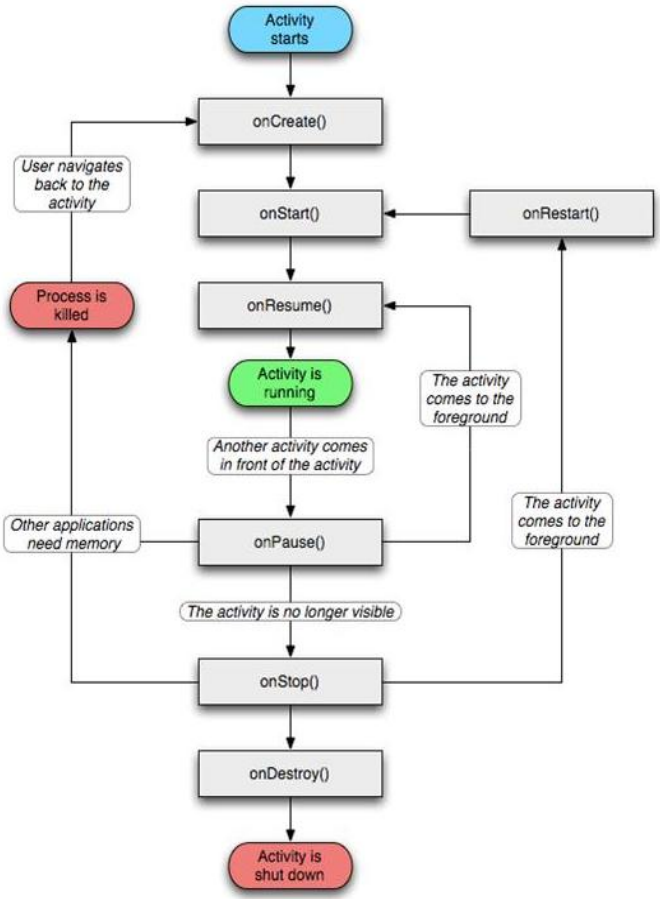

Figure 2. Activity Life cycle [2]

During its lifetime, each activity of an Android program can be in one of several states, which are managed by the system. However, you do get notified when the state is about to change through the on *XX*() method calls. You override these methods in your Activity class, and Android will call them at the appropriate time:

- onCreate(Bundle): This is called when the activity first starts up. You can use it to perform one-time initialization such as creating the user interface. onCreate( ) takes one parameter that program's persistent state, such as a database record being edited. is either null or some state information previously saved by the onSaveInstanceState() method.
- onStart( ): This indicates the activity is about to be displayed to the user.
- onResume( ): This is called when your activity can start interacting with the user. This is a good place to start animations and music.
- onPause( ): This runs when the activity is about to go into the background, usually because another activity has been launched in front of it. This is where you should save your
- onStop( ): This is called when your activity is no longer visible to the user and it won't be needed for a while. If memory is tight, onStop( ) may never be called (the system may simply terminate your process).
- onRestart( ): If this method is called, it indicates your activity is being redisplayed to the user from a stopped state.
- onDestroy( ): This is called right before your activity is destroyed. If memory is tight, onDestroy( ) may never be called (the system may simply terminate your process).
- onSaveInstanceState(Bundle): Android will call this method to allow the activity to save per-instance state, such as a cursor position within a text field. Usually you won't need to override it because the default implementation saves the state for all your user interface controls automatically.
- onRestoreInstanceState(Bundle): This is called when the activity is being reinitialized from a state previously saved by the onSaveInstanceState( ) method. The default implementation restores the state of your user interface.

Activities that are not running in the foreground may be stopped, or the Linux process that houses them may be killed at any time in order to make room for new activities. This will be a common occurrence, so it's important that your
application be designed from the beginning with this in mind. In some cases, the onPause( ) method may be the last
method called in your activity, so that's where you should save any data you want to keep around for next time.

3). Service

A Service is code that is long-lived and runs without a UI. A good example of this is a media player playing songs from a play list. In a media player application, there would probably be one or more activities that allow the user to choose songs and start playing them. However, the music playback itself should not be handled by an activity because the user will expect the music to keep playing even after navigating to a new screen. In this case, the media player activity could start a service using Context.startService() to run in the background to keep the music going. The system will then keep the music playback service running until it has finished. Note that you can connect to a service (and start it if it's not already running) with the Context.bindService() method. When connected to a service, you can communicate with it through an interface exposed by the service. For the music service, this might allow you to pause, rewind, etc.

4). Content Provider

Applications can store their data in files, an SQLite database, preferences or any other mechanism that makes sense. A content provider, however, is useful if you want your application's data to be shared with other applications. A content provider is a class that implements a standard set of methods to let other applications store and retrieve the type of data that is handled by that content provider.

5. Android User Interface

User Interfaces (UI) in Android can be built within two ways, by defining XML-Code or by writing Java-Code. Defining the GUI structure in XML is highly preferable, because as one knows from the Model-Viewer-Control principle that the UI should always be separated from the program-logic. Additionally adapting a program from one screen-resolution to another is a lot easier. Everything is well structured and can be expressed by tree-structures:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android=http://schemas.android.com/apk/res/android
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
         <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World" />
</LinearLayout>
```

Hierarchy of Screen Elements

The basic functional unit of an Android application is the activity-an object of the class android.app.Activity. An activity can do many things, but by itself it does not have a presence on the screen. To give your activity a screen presence and design its UI, you work with views and viewgroups-basic units of user interface expression on the Android platform.

Views

A view is an object extending the base class ndroid.view.View. It's a data structure whose properties store the layout and content for a specific rectangular area of the screen. A View object handles measuring, its layout, drawing, focus changes, scrolling, and key/gestures for the screen area it represents. The View class serves as a base class for all widgets - a set of fully implemented subclasses that draw interactive screen elements. Widgets handle their own measuring and drawing, so you can use them to build your UI more quickly. The list of widgets available includes i.e. TextView, EditText, Button, RadioButton, Checkbox, ScrollView, …

Viewgroups

A viewgroup is an object of class android.view.Viewgroup. As its name indicates, a viewgroup is a special type of view object whose function is to contain and manage a subordinate set of views and other viewgroups, Viewgroups let you add structure to your UI and build up complex screen elements that can be addressed as a single entity. The Viewgroup class serves as a base class for layouts - a set of fully implemented subclasses that provide common types of screen layout. The layouts give you a way to build a structure for a set of views.

A Tree-Structured UI

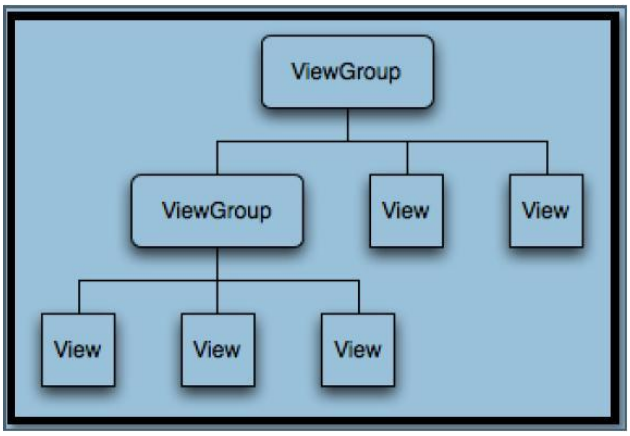

Figure 3. Android UI – Tree Structure [3]

On the Android platform, you define an Activity's UI using a tree of view and viewgroup nodes, as shown in the diagram below. The tree can be as simple or complex as you need to make it, and you can build it up using Android's set of predefined widgets and layouts or custom

view types that you create yourself. To attach the tree to the screen for rendering, your Activity calls its setContentView() method and passes a reference to the root node object. Once the Android system has the reference to the root node object, it can work directly with the node to invalidate, measure, and draw the tree. When your Activity becomes active and receives focus, the system notifies your activity and requests the root node to measure and draw the tree. The root node then requests that its child nodes draw themselves - in turn, each viewgroup node in the tree is responsible for drawing its direct children.

As mentioned previously, each view group has the responsibility of measuring its available space, laying out its children, and calling draw() on each child to let it render itself. The children may request a size and location in the parent, but the parent object has the final decision on where how big each child can be.

6. The androidmanifest.xml file

The AndroidManifest.xml is a required file for every Android application. It is located in the root folder of the application, and describes global values for your package, including the application components (activities, services, etc) that the package exposes to the 'outer world', what kind of data each of our Activities and co. can handle, and how they can be launched. An important thing to mention of this file are its so called IntentFilters. These filters describe where and when that activity can be started. When an activity (or the operating system) wants to perform an action such as open a Web page or open a contact picker screen, it creates an Intent object. This Intent-object can hold several information describing what you want to do, what data is needed to accomplish it and other bits of information. Android compares the information in an Intent object with the intent filter exposed by every application and finds the activity most appropriate to handle the data or action specified by the caller. If there is more than one application capable of handling that Intent, the user gets asked, which app he would prefer handling it. Besides declaring your application's Activities, Content Providers, Services, and Intent Receivers, you can also specify permissions in AndroidManifest.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="org.anddev.android.hello_android">
	<application android:icon="@drawable/icon">
		<activity android:name=".Hello_Android" android:label="@string/app_name">
			<intent-filter>
				<action android:name="android.intent.action.MAIN" />
				<category android:name="android.intent.category.LAUNCHER" />
			</intent-filter>
		</activity>
	</application>
</manifest>
```

- Almost every AndroidManifest.xml (as well as many other Android XML files) will include the namespace declaration (xmlns:android=http://schemas.android.com/apk/res/android) in its first element. This makes a variety of standard Android attributes available in the file, which will be used to supply most of the data for elements in that file.
- Almost every manifest includes a single <application> tag, which itself contains several tags describing Applications, IntentReceivers, etc… that are available in this application.
- If you want to make an Activity launchable directly through the user, you will need to make it support the MAIN action and LAUNCHER category.

7. Resources

Resources

Resources are external files (non-code files) that are used by your code and compiled into your application at build time. Android supports a number of different kinds of resource files, including XML, PNG, and JPEG files. The XML files have very different formats depending on what they describe. Resources are externalized from source code, and XML files are compiled

into a binary, fast loading format for efficiency reasons. Strings are compressed into a more efficient storage form.

List of resources
Resource-types and where to place them:
1. layout-files -> “/res/layout/”
2. images -> “/res/drawable/”
3. animations -> “/res/anim/”
4. styles, strings and arrays -> “/res/values/”
   Names do not have to be exactly like:
   - ‘arrays.xml’ to define arrays
   - ‘colors.xml’ to define colors
   - ‘dimens.xml’ to define dimensions
   - ‘strings.xml’ to define strings
   - ‘styles.xml’ to define style objects

5. raw files like mp3s or videos -> “/res/raw/”

R.java

A project's R.java is an auto-generated file indexing all the resources of your project. You use this class in your source code as a sort of short-hand way to refer to resources you've included in your project. This is particularly powerful with the code-completion features of IDEs like Eclipse because it lets you quickly and interactively locate the specific reference you're looking for. Additionally you gain compile-time safety that the resource you want to use really exists.

8. Conclusion

We have examined Android in a bottom-up fashion from its architecture to obtain an extensive overall understanding. We focused on the following general topics to clarify the design details of Android: architecture, installing and configuring steps to work with IDE, building blocks of Android including Activity, Intent, Service and Content Provider, Android User Interface including View, View Group and Tree structure UI, Android manifest and Resources including auto-generated R.java file. In order to understand how mobile application development can easily be understood from the core while working with Android, we depicted and summarized the key features of Android with a reasonable level-of-details in each topic.

References

[1] Open Handset Alliance, http://www.openhandsetalliance.com
[2] Android Developers Official Website, http://developer.android.com/
[3] Architecture of Android: http://www.android.com/about/videos.html#video=androidologyiarchitecture
[4] Android SDK Download, http://developer.android.com/sdk/index.html
[5] Eclipse IDE Download, http://www.eclipse.org/downloads/
[6] Android Development Tools plugin for the Eclipse IDE, http://developer.android.com/sdk/eclipse-adt.html
[7] Location to retrieve ADT for Eclipse IDE, https://dl-ssl.google.com/android/eclipse
[8] J2SE Download, http://www.oracle.com/technetwork/java/javase/downloads/index.html
[9] Lauren Darcey and Shane Conder, “Android Wireless Application Development”, Pearson Education, 2nd ed. (2011)
[10] Reto Meier, “Professional Android 2 Application Development”, Wiley India Pvt Ltd (2011)
[11] Mark L Murphy, “Beginning Android”, Wiley India Pvt Ltd (2009)
[12] Sayed Y Hashimi and Satya Komatineni, “Pro Android”, Wiley India Pvt Ltd (2009)