Volume 3, Issue 7, July 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Unveiling of Android Platform

A.Mallikarjuna
Department of Computer Science
S.V University, Tirupati, India

S.Madhuri
Department of Information Technology
JNTU, Hyderabad, India

Abstract: ***The Android platform was announced with the founding of the Open Handset Alliance, a consortium of 48 hardware, software, and telecom companies devoted to advancing open standards for mobile devices. Google has made most of the Android platform available under the Apache free-software and open source license. Android is a freely downloadable open source software stack for mobile devices that includes an operating system, middleware and key applications based on Linux and Java. Google developed Android collaboratively as part of the Open Handset Alliance, a group of more than 30 mobile and technology companies working to open up the mobile handset environment. Android's development kit supports many of the standard packages used by Jetty, and so, due to that fact and Jetty's modularity and lightweight footprint, it was possible to port Jetty to it so that it will be able to run on the Android platform. This paper on Android deals with the history of the Android, the early prototypes, basic building blocks of an android application and the features of the android.***

Keywords: Middleware,apache,consortium,prototypes,platform.

I. Introduction:

Android is a software platform and operating system for mobile devices, based on the Linux kernel, and developed by Google and later the Open Handset Alliance. It allows developers to write managed code in the Java language, controlling the device via Google-developed Java libraries. Applications written in C and other languages can be compiled to ARM native code and run, but this development path isn't officially supported by Google.

Android is available as open source. Google threw open the entire source code (including network and telephony stacks) that were not available previously, under an Apache license. Certain parts that relate to a specific hardware can't be made open and are not considered part of the Android platform. With Apache License, vendors are free to add proprietary extensions without submitting those back to the open source community. While Google's contributions to this platform are expected to remain open-sourced, the branches could explode using varieties of licenses.

Features of Android:

- **Handset layouts** Android can adapt to traditional smart phone
- layouts, as well other VGA, 2D, and 3D graphics libraries.
- **Storage** Android uses SQLite to store all its junk-- I mean, information.
- **Connectivity** Android supports a wide variety of technologies, including Bluetooth, WiFi, GSM/EDGE, and EV-DO.
- **Messaging** MMS and SMS are available for Android, as well as threaded text messaging. So you can send as many texties as you like.
- **Web Browser** Android comes pre-loaded with the Web Kit application. Remember, if you don't like it, you can always switch it out for something else later on thanks to the open source nature of the Google Android backend.
- **Java Virtual Machine** Software you write in Java can be compiled in Dalvik Byte codes (say that five times fast. I keep ending up with "Danish light bulb".) These can then be put into a Dalvik Virtual Machine. Basically more robust applications are supported than on some other Mobile Operating Systems.
- **Media Support** Android supports a wide range of audio, video, media, and still formats. MPEG-4, OGG, and AAC are just a few of these. Unfortunately the Media Player as its known right now is pretty basic, although more robust offerings on are the horizon from 3rd Party developers.
- **Additional Hardware Support** Got a touch screen you want to put to its full use? No problem. Android is capable of utilizing outside hardware like GPS, accelerometers, and all that other fun stuff.

Building blocks to an Android application:

There are four building blocks to an Android application:

- **Activity**
- **Broadcast Intent Receiver**
- **Service**
- **Content Provider**

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 and 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.

Intent and Intent Filters:

Android uses a special class called Intent to move from screen to screen. Intent describes what an application wants done. 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 URI. For example, to view contact information for a person, you would create intent with the VIEW action and the data set to a URI representing that person. There is a related class called an Intent Filter. While an intent is effectively a request to do something, an intent filter is a description of what intents an activity (or Broadcast Receiver, see below) is capable of handling. An activity that is able to display contact information for a person would publish an Intent Filter that said that it knows how to handle the action VIEW when applied to data representing a person. Activities publish their Intent Filters in the **AndroidManifest.xml** file.

. The new activity is informed of the intent, which causes it to be launched. The process of resolving intents happens at run time when start Activity 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 Intent Filter

II. BROADCAST INET RECEIVER:

You can use a Broadcast Receiver 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. Broadcast Receivers do not display a UI, although they may use the Notification Manager to alert the user if something interesting has happened. Broadcast Receivers are 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 BroadcastReceivers to be called; the system will start your application, if necessary, when a BroadcastReceiver is triggered. Applications can also send their own intent broadcasts to others with **Context.sendBroadcast ().**

III. 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.

IV. CONTENT PROVIDER:

Applications can store their data in files, an SQLite database, 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. Not every application needs to have all four, but your application will be written with some combination of these. All the components needed for android application should listed in an xml file called AndroidManifest.xml. This is an XML file where you declare the components of your application and what their capabilities and requirements are.

Storing, Retrieving and Exposing Data:

A typical desktop operating system provides a common file system that any application can use to store and read files that can be read by other applications. Android uses a different system on Android, all application data are private to that application. However, Android also provides a standard way for an application to expose its private data to other applications. This section describes the many ways that an application can store and retrieve data, expose its data to other applications, and also how you can request data from other applications that expose their data.

Android provides the following mechanisms for storing and retrieving data:

Preferences

A lightweight mechanism to store and retrieve key/value pairs of primitive data types. This is typically used to store application preferences.

Files

You can store your files on the device or on a removable storage medium. By default, other applications cannot access these files.

Databases

The Android APIs contain support for SQLite. Your application can create and use a private SQLite database. Each database is private to the package that creates it.

Content Providers

A content provider is a optional component of an application that exposes read/write access to an application's private data, subject to whatever restrictions it wants to impose. Content providers implement a standard request syntax for data, and a standard access mechanism for the returned data. Android supplies a number of content providers for standard data types, such as personal contacts.

Network

Don't forget that you can also use the network to store and retrieve data.

Security and Permissions in Android:

Android is a multi-process system, where each application (and parts of the system) runs in its own process. Most security between applications and the system is enforced at the process level through standard Linux facilities, such as user and group IDs that are assigned to applications. Additional finer-grained security features are provided through a "permission" mechanism that enforces restrictions on the specific operations that a particular process can perform, and per-URI permissions for granting ad-hoc access to specific pieces of data.

System Architecture:

A central design point of the Android security architecture is that no application, by default, has permission to perform any operations that would adversely impact other applications, the operating system, or the user. This includes reading or writing the user's private data such as contacts or e-mails, reading or writing another application's files, performing network access, keeping the device awake, etc.

An application's process is a secure sandbox. It can't disrupt other applications, except by explicitly declaring the permissions it needs for additional capabilities not provided by the basic sandbox. These permissions it requests can be handled by the operating in various ways, typically by automatically allowing or disallowing based on certificates or by prompting the user. The permissions required by an application are declared statically in that application, so they can be known up-front at install time and will not change after that.

Application Signing:

All Android applications (.apk files) must be signed with a certificate whose private key is held by their developer. This certificate identifies the author of the application. The certificate does *not* need to be signed by a certificate authority: it is perfectly allowable, and typical, for Android applications to use self-signed certificates. The certificate is used only to establish trust relationships between applications, not for wholesale control over whether an application can be installed. The most significant ways that signatures impact security is by determining who can access signature-based permissions and who can share user IDs.

User IDs and File Access:

Each Android package (.apk) file installed on the device is given its own unique Linux user ID, creating a sandbox for it and preventing it from touching other applications (or other applications from touching it). This user ID is assigned to it when the application is installed on the device, and remains constant for the duration of its life on that device.

Using Permissions:

A basic Android application has no permissions associated with it, meaning it can not do anything that would adversely impact the user experience or any data on the device. To make use of protected features of the device, you must include in your AndroidManifest.xml one or more **<uses-permission>** tags declaring the permissions that your application needs.

The permissions provided by the Android system can be found at **Manifest. permission**. Any application may also define and enforce its own permissions, so this is not a comprehensive list of all possible permissions.

A particular permission may be enforced at a number of places during your program's operation:

- At the time of a call into the system, to prevent an application from executing certain functions.
- When starting an activity, to prevent applications from launching activities of other applications.
- Both sending and receiving broadcasts, to control who can receive your broadcast or who can send a broadcast to you.
- When accessing and operating on a content provider.
- Binding or starting a service

Declaring and Enforcing Permissions:

To enforce your own permissions, you must first declare them in your AndroidManifest.xml using one or more **<permission>** tags.

The **<protection Level>** attribute is required, telling the system how the user is to be informed of applications requiring the permission, or who is allowed to hold that permission, as described in the linked documentation.

The **<permission Group>** attribute is optional, and only used to help the system display permissions to the user. You will usually want to set this to either a standard system group (listed in android.Manifest.permission_group) or in more rare cases to one defined by yourself. It is preferred to use an existing group, as this simplifies the permission UI shown to the user.

Note that both a label and description should be supplied for the permission. These are string resources that can be displayed to the user when they are viewing a list of permissions (android:label) or details on a single permission (android:description). The label should be short, a few words describing the key piece of functionality the permission is protecting. The description should be a couple sentences describing what the permission allows a holder to do. Our convention for the description is two sentences, the first describing the permission, the second warning the user of what bad things can happen if an application is granted the permission.

Applications Developed on Android Platforms:

- In September 2008, Motorola confirmed that it was working on hardware products that would run Android.
- Huawei Technologies is planning to launch smart phones that would run Android in Q1 2009.
- Lenovo is working on an Android-based mobile phone that supports the Chinese 3G TD-SCDMA standard.
- HTC is planning a "portfolio" of Android based phones to be released summer of 2009.
- Sony Ericsson is planning to release an Android based handset in the summer of 2009.
- Samsung plans to offer a phone based on Google's Android operating system in the second quarter of 2009.
- GiiNii Movit Mini is a Internet device based on Google's Android operating system

V. Conclusion:

Finally we concluded that the Androids platform which has developed by Google is going to play major role in Mobile applications because as it is an open source and it is also easy to develop mobile applications using Android as because in order to develop these applications all the APIs are available and these APIs are as same as java APIs which are easy to understand.

References:

[1] Licenses *Android Open Source Project*. Open Handset Alliance. http://source.android.com/license. Retrieved on 22 October 2008.

[2] Open Handset Alliance (5 November 2007). Industry Leaders Announce Open Platform for Mobile Devices. Press release. http://www.openhandsetalliance.com/press_110507.html. Retrieved on 5 November 2007.

[3] Google's Android parts ways with Java industry group. http://www.news.com/8301-13580_3-9815495-39.html. General Android http://code.google.com/android/kb/general.html#c. Retrieved on 29 August 2008.

[4] Native C application for Android http://benno.id.au/blog/2007/11/13/android-native-apps.

