imaginea white paper

Building Mobile Android™ Applications

Even though Android was created for handsets, there is a great opportunity for developing other innovative devices on the Android platform with significant optimizations and additions required to optimize Android for other connected devices. With no licensing and lengthy application review process, the Android Market is growing faster than any App Store. Having a presence in this growing segment and tapping the growing opportunities must be high on the list of any enterprise targeting the application market for mobile phones and other connected devices. Building applications for Android addressing the broadest range of devices requires a specialist, such as Imaginea.

Contents

What is Android?

Android™ means different things to different people. It is easy to think of Android[1] as yet another operating system for high-end mobile phones. It is really a software platform, rather than just an OS, that is beingutilized in a wide range of devices like Tablet PCs, car infotainment systems and even regular PCs. Android is a software stack for mobile devices that includes an operating system, middleware and key consumer applications that have ubiquitous utility. While the initial focus was on the consumer segment, applications targeting businesses have already started appearing on the Open Android Market.

The Android SDK provides the tools and APIs necessary to develop applications on the Android platform using the Java programming language. Android is open source and a majority of the source is licensed under Apache2, allowing adopters, vendors to add additional proprietary value to the Android source without source distribution requirements.

Android Opportunity

The Android story began in 2005 when Google™ acquired Android Inc. The Android distribution was announced along with the formation of the Open Handset Alliance in 2007, with a common goal of fostering innovation on mobile devices and giving consumers a far better user experience than what was available on the mobile platforms of the day. Soon Google released most of the code as opensource. Even though Android was created for handsets, many developers began to see a great opportunity for developing other innovative devices on the Android platform with significant optimizations and additions required to optimize Android for other connected devices. Also, to support the growing ecosystem of devices, a host of applications had to be developed and supported on various types and sizes of devices. According to the NPD Group, unit sales for Android OS smart phones ranked second among all smart phone OS handsets sold in the USA in the first quarter of 2010, while BlackBerry™ OS and the Apple™iOS™were first and third respectively. With no licensing and lengthy application review process, the Android Market is growing faster than any App Store. Having a presence in this growing segment and tapping the growing opportunities must be high on the list of any enterprise targeting the application market for mobile phones and other connected devices.

Android Architecture

Android is a complete software stack for mobile devices as shown in Figure 1. The Linux Kernel specially tuned for a mobile environment, forms the base of the Android software. The Android kernel supports its own Power Management (on top of the standard Linux Power Management) designed on the premise that the CPU shouldn't consume power if no applications or services require power. Android requires that applications request CPU resources with "wake locks" through the Android application framework and native Linux libraries. If there are no active wake locks, Android will shut down the CPU.

Source: http://developer.Android.com/guide/basics/what-is-Android.html
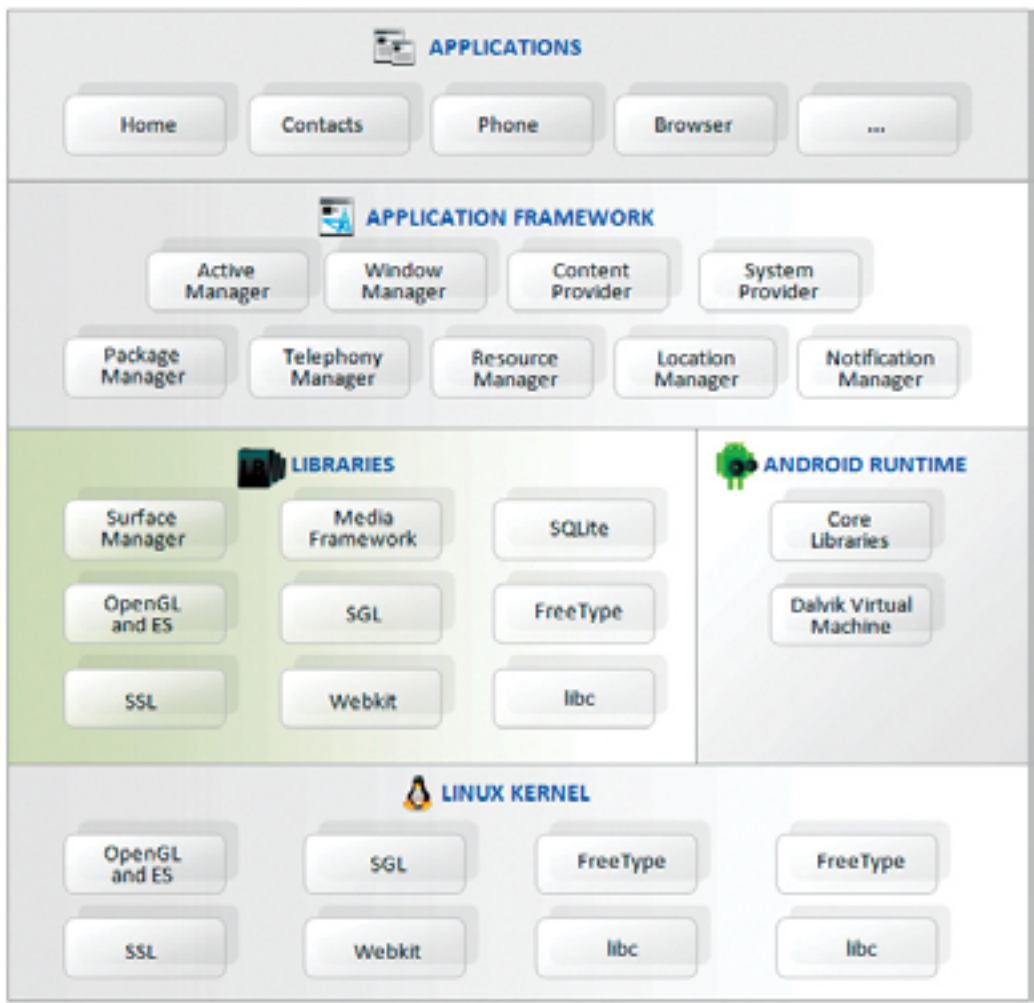
Figure 1:
High-level look at
the Android System
Architecture

Android incorporates OS features like efficient shared memory with the Binder mechanism, pre-emptive multi-tasking, UNIX user identifiers (UIDs), group identifiers (GID) and file permissions. The security model is more like a multi-user system as compared to the sandbox found on the J2ME or Blackberry platforms. Android applications are in individual silos unlike a desktop computer environment where user applications run with the same UID. Android applications run in separate processes under distinct UIDs and group identifiers (GID).

Multiple applications signed with the same certificate can run in the same process, but this has to be specified explicitly. Typically, applications can neither Read nor Write each other's data or code, and applications can only share data between themselves explicitly. The table below shows the output of the "ls command" for few of the standard applications on Android. This shows how some applications share a common UID and the remaining applications have an unique UID assigned by Android.

Table 1.1: Output of the 'ls' Command for some Standard Android Applications

Permissions	UID	GID	Date	Application Private Directory
drwxr-xr-x	app_40	app_40	2008-10-18 02:33	com.Android.voicedialer
drwxr-xr-x	app_38	app_38	2008-10-18 02:32	com.Android.bugreport
drwxr-xr-x	app_49	app_49	2009-06-06 14:32	com.Android.spare_parts
drwxr-xr-x	app_21	app_21	2008-10-18 02:33	com.Android.vending
drwxr-xr-x	system	system	2008-09-05 22:43	com.google.Android.systemupdater
drwxr-xr-x	radio	radio	2008-09-04 00:49	com.Android.providers.telephony
drwxr-xr-x	radio	radio	2010-02-17 13:09	com.Android.phone
drwxr-xr-x	app_55	app_55	2010-02-21 22:59	org.connectbot

On top of custom Android Linux kernel are a set of libraries including bionic (libc), multimedia support for audio and video, graphics, openGLand SQLite— a lightweight database.

Dalvik Virtual Machine (VM) is a key component of an Android Runtime system and is intended to run on a variety of target platforms. It is designed to be instantiated multiple times – each application has its own private copy running in a process. The Dalvik VM makes full use of Linux for memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM executes files in the Dalvik Executable (.dex) format, which is optimized for minimal memory footprint. The Application Framework provides many higher-level services to applications in the form of Java APIs, which are part of Android SDK.

At the top of the Android software stack are applications. A key Android capability is the loose coupling and sharing of functionality, where in each application can make use of other system applications or even the functionality of regular applications to accomplish critical tasks in the user workflow. Every application can export functionality for use by other applications in the system, thus promoting software re-use and a consistent user experience. For example, an email application can expose the ability to create and send emails to other applications, and the user has the ability to pick any email application installed on his device with a consistent user interface. Users can directly call, email or ftp to a server while browsing in a web browser app, if Dialer, Email and FTP applications are installed on their device and also pick one application from several installed applications, providing similar service. The figure below shows

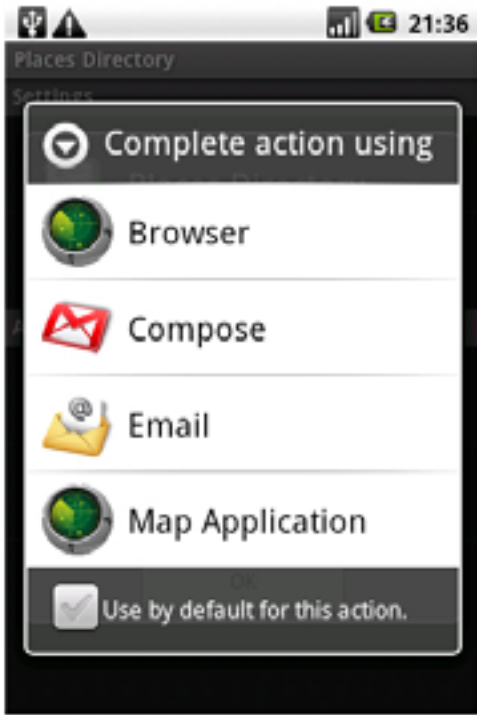the App Chooser UI to accomplish a particular action in a workflow.

Figure 2: The Android App Chooser UI

All Android applications have the same status in a given system, though the applications with system privileges usually have more privileges. System applications either define special system permissions, which they enforce for security, or they have the same UID by virtue of being signed with the device manufacturer's certificate during the system-build that the device ships with.

Application Development

The Environment

Eclipse is the Android development environment recommended by Google. The Android Development Tools (ADT) is a plugin for the Eclipse IDE designed to give a powerful, integrated environment to build Android applications. However, developers are not restricted to any particular IDE if they prefer to use command line tools.

ADT extends the capabilities of Eclipse and helps to quickly set up new Android projects, create an application UI, add components based on the Android Framework API, debug applications using the Android SDK tools and export signed (or unsigned) APKs (Android Package Files) in order to distribute an application.

The different Android platforms are available as a downloadable component for the Android SDK. The platform includes a fully compliant Android library and system image, as well as a set of emulator skins, sample applications, and more. Developers need to define their target platform configuration by specifying an Android Virtual Device. They can then execute code on either the host-based emulator or a real device, which is normally connected via USB.

Development

An Android application consists of a number of resources which are bundled into an archive – an Android package file called APK. Programs are

generally written in Java, built using standard Java tools, and then the output file is processed to generate specific code for the Dalvik VM. Each application runs in its own process – an instantiation of the Dalvik VM, which protects its code and data from other applications. There are several mechanisms for applications to transfer, exchange, and share data. Many developers will also want their C/C++ applications to run on an Android-based device, for which Android Native Development Kit (NDK) is provided. The Android Market has a built-in filtering mechanism using which it filters applications to display them only on devices for which the native code was compiled.

There are four types of application components: Activities, Services, Broadcast Receivers and Content Providers.

- An Activity allows applications to call each other, and is responsible for setting the layout for the User interface. It allows reusing features to accomplish any given task in a complex workflow. An application may incorporate a number of activities. A Launcher activity is an activity that is set as the default, which means that it can be directly launched by the user from the Home screen on his device.

- A Service is a long running background task which runs in the same process or a different process. An example of a service might be a Download Service that downloads music or applications while the user performs other tasks.

- Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, it may be useful for the application to know when the system boot completed, so that it can start a Service in the background, or a picture has been taken.

- Content Provider is a mechanism to allow applications to share raw data. This can be implemented to share SQL data, images, videos etc. Common Content Providers to access resources are provided by the SDK.

When developing an Android application, developers need to describe it to the system and this is achieved by means of a manifest file. This is an XML file called AndroidManifest.xml and each application component is specified in the manifest file for the component to be recognized and used in the system. Applications and Services running on the phone can be viewed and managed using the Settings application. A running services display is shown below.
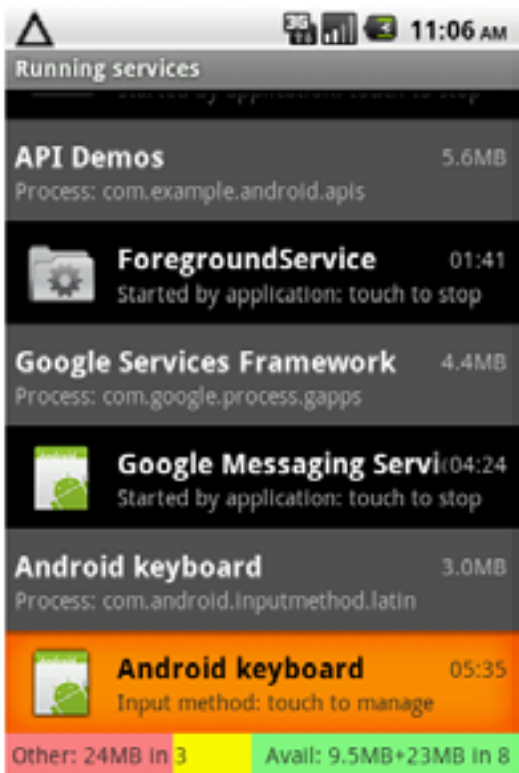
Figure 3: The Android 'Running Services' Display

When an Android application has to obtain some functionality from another application or from the system, it can issue Intents. Intents are a mechanism for moving data between Android processes and are at the core of much of Android's Inter Process Communication (IPC). For example, the intent to start an email application can contain the email address of "To", "cc ", subject and even the body. Intents don't enforce security policy themselves, but the framework and/or the receiver is usually responsible for enforcing any security. Almost all Android IPC is actually implemented through

Binder, although most of the time this is hidden from users using higher level abstractions. This is an asynchronous message used to activate an activity, service, or broadcast receiver. The specific action and location of data are included for an activity or service.

An 'Intent' may include the specific activity required. It can be more generalized where the request is resolved dynamically by the system and the user is presented with a list of applications handling that Intent. This mechanism is governed by Intent Filters. These filters specify the types of Intents that the activities and services of an application can handle. Developers need to carefully ensure that sensitive data is not transferred using Intents, when setting up permissions or when the Broadcast Intents are sent, so that rogue applications do not misuse the data.

Android requires developers to sign their code before running on production devices. Android code signing usually uses self-signed certificates, which developers can generate on their own. Applications signed with the same key (and therefore by the same developer) can also ask to run with the same UID by specifying the "sharedId" in the manifest file. This allows developers to upgrade or patch their software easily, including copying data from existing versions or across applications. Code signing is enabled through the ADT or by the ANT tool target for those wanting to automate the build process.

Android Top 10 Best Practices

1. Share common functionality that you want exposed and do not declare Intents Filters explicitly for functionality you don't want exposed. If you are planning multiple applications that need to share data, declare a "sharedId" in the manifest explicitly so that all these applications can access each other's data with the same UID. Specifying the "sharedId" property later does not help as it prevents even an upgrade of the same app to access its data. Proper planning needs to be done at the outset before you decide to distribute your application.

2. Develop for the least common denominator when focusing on target platforms to reach out to the broadest audience. Do not make assumptions about device hardware (touch etc.). Allow a functionality to be accessed by various means – menus, context menus, navigation toolbars and, where applicable, voice and hand gestures.

3. Handle screen orientation changes-Applications can override the user's setting, and can always rotate with the device or remain fixed to one orientation. If the device has a physical keyboard, applications can also change orientation when the physical keyboard is opened or closed. This means that your user interface must be prepared for an orientation change, unless your application is designed to always display in portrait or landscape mode. Develop layouts for screen orientation changes if required.

4. Use layouts that adapt to multiple devices. Many Android devices have been released and many more are expected to be released this year. The user interface layout needs to adapt to screens of various physical sizes, densities and resolutions. As far as possible, try to develop your layouts so they adapt to multiple devices.

5. Follow the native platform's UI guidelines and best practices instead of trying to, say, implement an iPhone™ look and feel on the Android, which will confuse Android users.

6. Use Services and Wake Locks carefully. When a Service is done with processing a long running job, it should stop by itself. Similarly a Wake Lock should be released immediately to avoid draining the battery.

7. Starting with Froyo release, applications can be installed on the SD Card, saving internal memory. Carefully review whether your application can be installed externally allowing the user to install more applications internally. Back up user preferences and files using the Android Cloud services so that user can

recover the data in the event of data loss or device theft.

8. Localization- Localize your app in different languages to reach out to the entire world.

9. Test Automation- Develop test cases continuously and measure your test coverage with Emma instrumentation coverage before the code gets out of hand with different SDK version and device types. Always test for different devices types, resolutions and screen orientation changes. Use Monkey Tool for proper UI workout.

10. Build Process – Do a clean build with Apache Ant tool for production and test builds and Hudson for continuous integration.

Comparison with iPhone and Blackberry

Each operating system has its own strengths and weaknesses, and has something to offer to developers. Developing applications in the iPhone environment is easier than on Android and Blackberry, thanks to the uniform screen size of the iPhone. This situation has changed with the advent of the iPad™, and the iPhone OS has been rebranded as iOS. Blackberry is primarily a Windows-only development environment, while the iPhone requires MacOS. On the other hand, developers can build applications on Windows, Linux and MacOS platforms when using Android. However, an application for BlackBerry, Android and iOS devices has to be tweaked to fit different screen sizes.

BlackBerry's OS can present significant challenges to developers. It can be difficult to make a single application that is interoperable with the wide variety of BlackBerry devices. BlackBerry is not the easiest of Operating Systems to develop applications, since there are so many different versions of the OS. Something that works on the Tour isn't guaranteed to work on the Bold.

On the other hand, the development model for Android is consistent. The same code runs on all the Android devices. The device dependencies are isolated into xml configuration files or folder structures for resource loading. Applications can choose to support multiple screens. However, if the application does not support multiple screens the platform still performs best-effort rendering of the application. While it get can be quite a challenge to handle the various devices, Android allows developers to target different device

types in a phased manner, if required. The inbuilt filtering mechanism of the Android Market shows the application only on the devices for which it has been targeted. Other features, unique to Android include, widget support, the provision given for a user to send crash reports to developers and developer access through the Market interface.

Apple iPhone SDK 3.2 supports the development of Universal applications. A Universal app is optimized to run on all iPhone OS devices. It is essentially an iPhone app and an iPad app built as a single binary. But the UI elements for various device layouts are stored in a format called 'nib,' which is hard to read, as opposed to the Android xml layouts. A number of APIs need to be carefully used to work across IPhone and IPad. Conditional coding is essential for resource loading some of the Cocoa Touch classes, functions and methods, new APIs in existing frameworks, and also for detecting certain hardware capabilities.

While cross device application programming frameworks such as PhoneGap (based on JavaScript, HTML and CSS) and Rhodes (based on Ruby) can be used to build for Android, Blackberry and iPhone, they do not offer the full capabilities of a native platform. They however offer an alternative to developing with device specific APIs.

The limited guidelines or lack of visibility into the App approval process makes it harder to get applications approved on the Apple App Store. On the other hand, the guidelines for Blackberry are much more detailed and easier to get approval. Because Android is open source, any developer can access its source code and create apps without getting a license from Google. It is extremely easy to get application on the Android Market, as Google does not act as a gatekeeper. Google allows all apps onto the store and only removes inappropriate apps if they are found violating the distribution agreement.

Table 1.2: Comparison of the Android, iPhone and Blackberry Platforms

Permissions	Android	iPhone	Blackberry
Application Sandboxing for Security	Yes	No	No
Ease of Development for different screen sizes Conditional Coding required for screen size support	Yes 8/10 No Device dependencies can be isolated into xml layout configuration files or folder structures for resource loading	Yes 8/10 Yes The jury is still out on this one, not too many devices to target currently.	No 3/10 Yes. May not be possible across devices running different OS versions.
Language	Java, Android SDK, Android Native Development Kit (NDK)	Objective-C	J2ME, Native Blackberry SDK.
Development Support on Windows, Mac, Linux.	Yes	No Mac Only	Yes Windows Only
UI Builder	No	Yes	No (J2ME based needs to be customized)
Eclipse JDT	Yes	No |(Custom Tools –XCode)	Yes
Garbage Collection	Yes Automatic	Yes Automatic, Manual Allocation and De-allocation of objects allowed.	Yes Automatic
Memory Profiler and Heap Analysis	Yes 8/10	Yes 10/10	Yes 8/10
Background processes	Yes	Yes Background processing support starting with iOS 4.	Yes
Push Notifications	No Cloud to Device notifications available starting only with 2.2	Yes	Yes
Widget Support	Yes	No	No
App Store Support for Paid applications in multiple countries	Yes Limited to very few countries.	Yes	Yes
Easy to get app approved on App Store	Yes Open	No No Visibility into approval process.	Yes Detailed Guidelines available
Ability to run cross-OS application programming frameworks such as PhoneGap (based on Javascript, HTML and CSS) and Rhodes (based on Ruby)	Yes	Yes	Yes

Android Development Challenges

The Android deployment was initially focused on mobile handsets, primarily for the consumer market, which to a great extent, is still the main focus. The software IP and development tools were designed and configured by Google with this target market in mind. The potential for Android in other markets such as Consumer, Telecom, Automotive, Medical and Home applications, is enormous. Android lends itself to some specific applications:

- Android offers a good mix of capabilities for digital video applications. Google TV, an effort in this direction, is an open platform for TV-related devices that brings together the best of TV and the best of the web to deliver the premier entertainment experience. It is built on Android and runs the Google Chrome web browser.

- In cars and other road vehicles, Android appears to be an ideal choice for in-vehicle infotainment devices, such as satellite navigation.

- New home and industrial electronic devices appearing in the market are an ideal fit for Android. Here again, connectivity and a good user interface are of prime importance.

However, there are challenges in moving away from mobile handsets. With reference to the Android system architecture (Figure 1), the requirements for Android development system can be identified as:

- Preparing Android Linux Kernel for a particular hardware target, which requires the careful application of numerous patches and industry-specific drivers and creation of new drivers for new device peripherals.

- Porting and optimization of libraries for different targets.

- Enterprises need support with writing new applications in Java for Android, and for allowing new or existing C/C++ applications to run on their new Android-based device.

- Applications need to interact with Services hosted on the Cloud, or even within an Enterprise. Google's Backup services for simple files and user preferences are already available on the Cloud for Android devices. Several services such as Printing are being moved to the Cloud. An example being Google Print Services.

Development System for Android-based devices from Imaginea

The Development System for Android-based Devices from Imaginea can address all the goals, mentioned above. Imaginea's approach to the Android market has these key elements helping prepare the Android Linux Kernel for a particular hardware, porting and optimizing libraries for different systems, contributing to the improvement of existing open source tools and/or creating new development tools, developing software IP and providing professional services. At the core of Imaginea's approach is a clear commitment to the needs of the customer, enabling them to leverage open software to address their specific requirements. There is support for customization of the application framework for device manufacturers. The goal is to create reproducible, high quality code to deploy Android into multiple application domains and to deploy Android applications onto multiple Android devices, leveraging Imaginea's Android and Cloud Computing expertise and experience that help customers get products and solutions faster to the market.

Imaginea can support enterprises throughout their entire product life cycle, from product requirements all the way through deployment, with complete Android services, including:

- Application Development

- Application Framework and System Applications customizations.

- Integration of product or industry-specific middleware

- Functional testing, stress and performance testing, and benchmarking

Conclusion

Android is a ground-breaking technology, which was introduced initially on mobile handsets, but has much wider potential. There are challenges in the application of Android to other types of device and domains. Building applications for the Android addressing the broadest range of devices requires a specialist, who can understand all these aspects and deliver a world class application. If you want to build an application that leverages our expertise, please get in touch with us at sales@imaginea.com

References

Android:

http://developer.Android.com

http://www.openhandsetalliance.com/Android_overview.html

http://source.Android.com/porting/power_management.html

http://code.google.com/events/io/2010/sessions.html

Security Architecture

http://developer.Android.com/guide/topics/security/security.html

iPhone Universal App development

http://devimages.apple.com/iphone/resources/introductiontouniversalapps.pdf

imaginea