TEAM MEMBERS

Martin Jantscher – FH Joanneum

Mohammed Talhaoui – Artesis Hogeschool Antwerpen

Denis De Vos – Haute École Provinciale de Mons-Borinage-Centre

Ivan Cunha – Escola Superior de Comunicação Social

Artur Roszcyk – Wyzsza Szkola Informatyki

Mikhail Datsyuk – Central Ostrobothnia

1 Introduction

What is Android?
Android is a software platform and operating system for mobile devices, based on the Linux kernel, 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 is not officially supported by Google.

The unveiling of the Android platform on 5 November 2007 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 released most of the Android code under the Apache license, a free-software and open source license.

What is The Open Handset Alliance?
Open Handset Alliance, is a consortium of several companies which include Google, HTC, Intel, Motorola, Qualcomm, T-Mobile, Sprint Nextel and NVIDIA, ...
These companies which aim to develop technologies that will significantly lower the cost of developing and distributing mobile devices and services. The Android platform is the first step in this direction -- a fully integrated mobile "software stack" that consists of an operating system, middleware, user-friendly interface and applications.

License
Android is under version 2 of the Apache Software License (ASL).
The Apache license allows manufacturers and mobile operators to innovate using the platform without the requirement to contribute those innovations back to the open source community.

2 Platform

2.1 Hardware

First and foremost, Android is a software stack for mobile devices. This means that high on the list of priorities is the preservation of battery power and the efficient management of limited memory resources. There are five distinct layers to the Android system stack:

- The Acorn RISC Machine (ARM) Linux core forms the solid base upon which all the other layers stand. Linux is a proven technology that is highly reliable, and the ARM processor family is known for high performance on very low power requirements.

- The libraries provide the reusable and sharable low-level code for basic functions such as *codecs* — software for coding and decoding digital sound and video — functions for the presentation of rich graphics on a small displays, secure shell support for encrypted TCP/IP traffic into the cloud, as well as component support for Web browsing (WebKit), SQL database functionality (SQLite), and standard C library functionality you would expect in a Linux system.

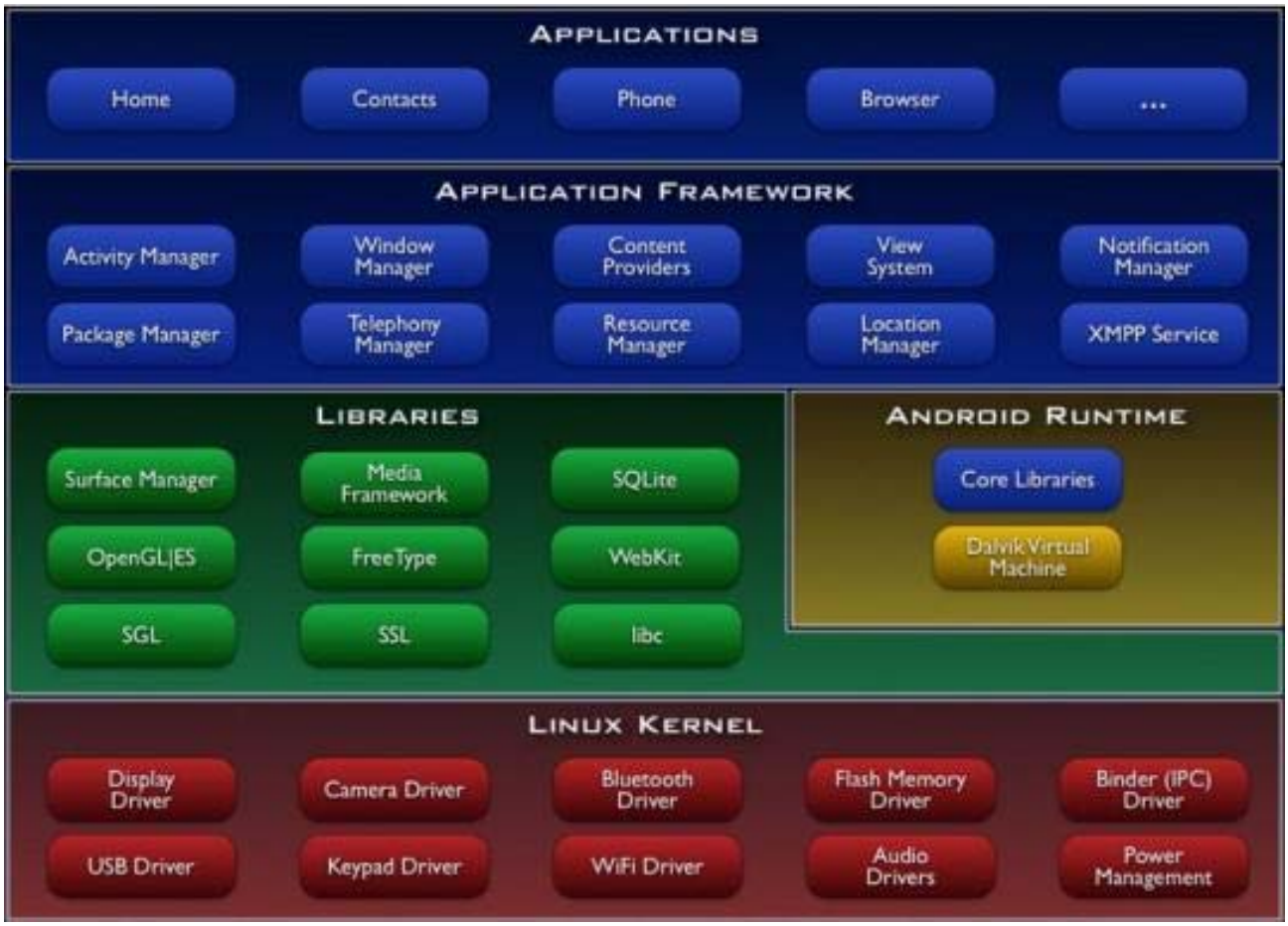
- The Dalvik run-time byte-code interpreter, which strongly resembles the Java™ language byte-code interpreter, adds a few distinct features that uniquely define the security and power-preserving model of Android. Every application currently running, for example, has its own user ID and its own copy of the interpreter running to strictly separate processes for security and reliability.

- The Android application framework enables you to use and replace components as you see fit. These high-level Java classes are tightly integrated components that define the Android API.

- The Android core applications include the WebKit browser, Google calendar, Gmail, Maps application, SMS messenger, and a standard e-mail client, among others. Android applications are written in the Java programming language, and you can download many more from the Android market on the fly.

Android is not a single piece of hardware; it's a complete, end-to-end software platform that can be adapted to work on any number of hardware configurations. Everything is there, from the bootloader all the way up to the applications. And with an Android device already on the market, it has proven that it has what it takes to truly compete in the mobile arena.

2.2 Operating System(s)



Android uses Linux for its device drivers, memory management, process management, and networking. However you will never be programming to this layer directly.

The next level up contains the Android native libraries. They are all written in C/C++ internally, but you'll be calling them through Java interfaces. In this layer you can find the Surface Manager (for compositing windows), 2D and 3D graphics, Media codecs (MPEG-4, H.264, MP3, etc.), the SQL database (SQLite), and a native web browser engine (WebKit).

Next is the Android runtime, including the Dalvik Virtual Machine. Dalvik runs dex files, which are converted at compile time from standard class and jar files. Dex files are more compact and efficient than class files, an important consideration for the limited memory and battery powered devices that Android targets.

The core Java libraries are also part of the Android runtime. They are written in Java, as is everything above this layer. Here, Android provides a substantial subset of the Java 5 Standard Edition packages, including Collections, I/O, and so forth.

The next level up is the Application Framework layer. Parts of this toolkit are provided by Google, and parts are extensions or services that you write. The most important component of

the framework is the Activity Manager, which manages the life cycle of applications and a common “back-stack” for user navigation.

Finally, the top layer is the Applications layer. Most of your code will live here, along side built-in applications such as the Phone and Web Browser

2.3 Network Connectivity

It supports wireless communications using:

- GSM mobile-phone technology
- 3G
- Edge
- 802.11 Wi-Fi networks

2.4 Security

Android is a multi-process system, in which 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.

Security 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.

2.5 Performance

Devices hosting Android applications have limited capabilities. That's why code should be efficient, avoid all unnecessary memory allocations, method calls (it takes a lot of time) and so on.

In order to make our applications working fast on a mobile device we need to leave back some habits, good from OOP point of view. In a mobile device we are not able to make a full model of reality what we want to operate on.

Few things to remember:

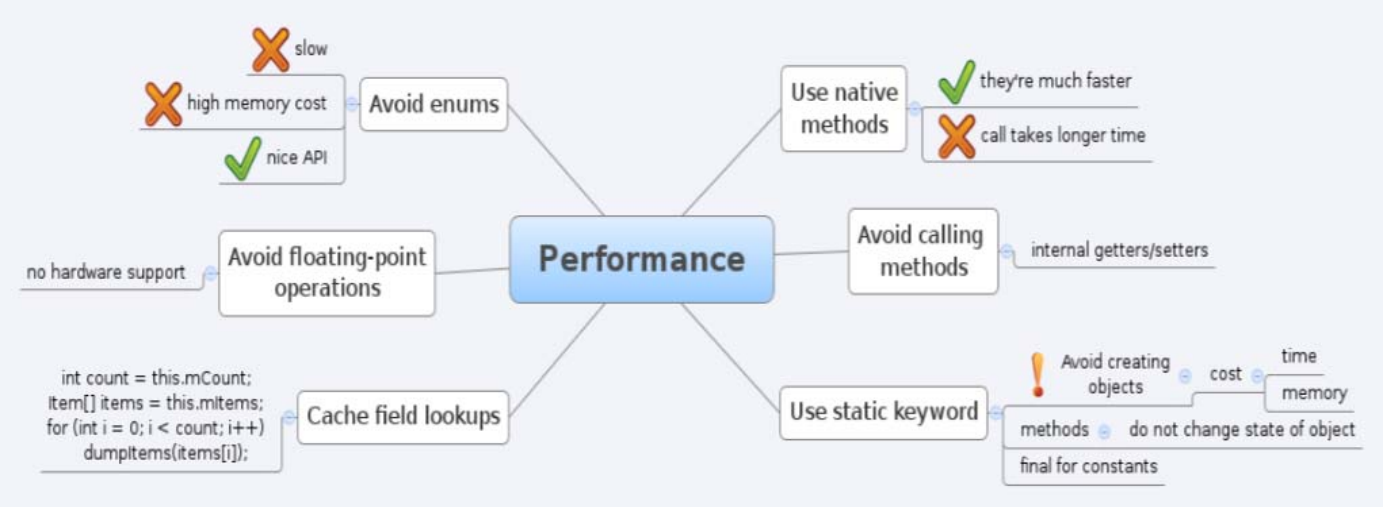
- **avoid object instantiation** - create objects only if it is really necessary, because it costs time and memory. More instances means more-frequent garbage collection what lowers user-experience (freezes).

- **use native built-in methods** - they're written in C/C++ what makes them faster about 10-100 times than implemented JAVA code (loops etc.). However note that calling native method is more expensive then calling implemented one.

- **virtual over interface** - in conventional programming it is usual to declare variables as interfaces, i.e.:
  Map myMap1 = new HashMap();
  It is not good for embedded applications. Calling a method from interfaces takes 2 times more time than in normal way: HashMap myMap2 = new HashMap();

- **static over virtual** - declare methods static if they do not need access to the object's fields. It can be called faster, because it doesn't require a virtual method table indirection. It's also good practice, because you can tell from the method signature that calling the method can't alter the object's state.

- **cache field lookups,** because accessing object fields is lower than local variables. The same situation is with methods - i.e. by for-statements, you should cache size() method if it is possible:



2.6 Future possibilities

Android sits alongside a new wave of mobile operating systems designed for increasingly powerful mobile hardware. Windows Mobile and Apple's iPhone now provide a richer, simplified development environment for mobile applications. However, unlike Android, they're built on proprietary operating systems that often prioritize native applications over those created by third parties and restrict communication among applications and native phone data. Android offers new possibilities for mobile applications by offering an open development environment built on an open source Linux kernel. Hardware access is available to all applications through a series of API libraries, and application interaction, while carefully controlled, is fully supported.

In Android, all applications have equal standing. Third-party and native Android applications are written using the same APIs and are executed on the same run time. Users can remove and replace any native application with a third-party developer alternative; even the dialer and home screens can be replaced.

- Google Android Sales to Overtake iPhone in 2012
- The OHA is committed to make their vision a reality: to deploy the Android platform for every mobile operator, handset manufacturers and developers to build innovative devices
- Intel doesn't want to lose ownership of the netbook market, so they need to prepare for anything, including Android
- Fujitsu launched an initiative to offer consulting and engineering expertise to help run Android on embedded hardware, which aside from cellphones, mobile internet devices, and portable media players, could include GPS devices, thin-client computers and set-top boxes.
- More Android devices are coming and some will push the envelope even further

3 Software Development

3.1 Development Requirements

Developing an application on the Android platform requires the following:

- knowledge of programming in Java
- knowledge of XML (optional but recommended)
- Android SDK (requires x86 OS like Windows, Linux, Mac; JDK version >=5)
- Eclipse IDE (at least version 3.3) with Android Development Tools (ADT) plug-in (optional but recommended)
- Android powered smartphone (optional for testing purposes)

Notice that JavaME and JavaSE applications aren't runnable on Android as the class libraries as well as the generated bytecode are different.

3.2 IDE and Tools

Android SDK
Beside of the actual java class library the Android SDK (latest version 1.1r1) contains all the tools that are necessary to build an Android application. Typically every Android SDK version consists of:

Developer Tools
As already mentioned above the SDK comes with a bunch of tools that relieve the creation of an Android app. In the following only the most important tools are described:

- aapt - Android Asset Packaging Tool
  Creates *.apk-files which contain all the resources as well as the program itself. Those ZIP-format based files can be transferred to and installed on an Android phone or the emulator.

- adb – Android Debug Bridge
  The main purpose of this tool is to set up connections to a real Android device or an Android emulator instance in order to transfer and install (apk)-files on it. With adb the developer also has the possibility to remote control the devices shell.

- dx – Dalvik Cross-Assembler
  The dx is used for merging and converting Java-Standard-ByteCode Classes (*.class) into one single binary file (*.dex) that can be executed by the Dalvik VM. These *.dex-files are subject to be put into an *.apk-file together with resource files.

- ddms - Dalvik Debug Monitor Service
  This tool provides port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process and radio state information, incoming call and SMS spoofing, location data spoofing, and more.

Emulator and System Images

The SDK also contains an emulator that is able to simulate almost all the functionality of an actual Android device. This is achieved by booting so called system images which represent the Android OS with the whole software stack of a real Android device.

Documentation, Sample Code

Of course the SDK also provides the developer with a comprehensive documentation which consists not only of an API reference but also of programming guidelines as well as detailed descriptions for the several tools. There are also a few code examples that will allow understanding the typical workflow of an Android application right away.

IDE Support

Although it is possible to develop Android apps with every modern IDE Google recommends doing so is by using the Eclipse IDE with a special plug-in called ADT (Android Development Tools). The ADT makes use of all the Dev Tools that come with the SDK and therefore supports and simplifies all the steps from assembling the classes over packaging and signing to running the final application on the emulator.

The ADT is not just speeding up the testing process but also relieves the developers work in terms of UI creation and application description. For that reason the ADT offers the developer graphical representations of what has otherwise have to be written in XML.

3.3 Programming language(s)

The officially supported programming language on the Android platform is Java. It is also recommended to have some knowledge of XML as the descriptor file as well as the user interface of an application is based on that.

As the Linux kernel of the Android platform is based upon an ARM processor architecture it would also be possible to write code in C or other languages and compile it to ARM native code.

4 Overall Evaluation

4.1 Advantages

There are a host of advantages that Google's Android will derive from being an **open source software**. Some of the advantages include:

- The ability for anyone to customize the Google Android platform will open up the applications playing field to small and new players who lack the financial muscle to negotiate with wireless carriers like AT&T and Orange.

- The consumer will benefit from having a wide range of mobile applications to choose from since the monopoly will be broken by Google Android.

- Although this will depend on the carrier, one will be able to customize a mobile phones using Google Android platform like never before, right down to the screen.

- Features like weather details, opening screen, live RSS feeds and even the icons on the opening screen will be able to be customized.

- In addition, as a result of many mobile phones carrying Google Android, companies will come up with such innovative products like the location – aware services that will provide users with any information they might be in need of.

- This information could include knowing the location of a nearby convenience store or filling station. In addition the entertainment functionalities will be taken a notch higher by Google Android being able to offer online real time multiplayer games.

4.2 Limitations

Bluetooth limitations

Google Talk functions and only the simplest implementation of Bluetooth. It'll work with Bluetooth headsets but that's about it; no Bluetooth stereo, no contacts exchange, no modem pairing and no using wireless keyboards.
Android uses a non-standard jvm: there is no guarantee that the same software will run on multiple devices

Firefox Mobile isn't coming to Android because of Android Limitations

Fennec won't play nice with Android Market because apps in Android Market need to be programmed with a custom form of Java to run on Android. Mozilla and the Fennec peeps won't have that and won't be releasing any form of Firefox until Google amends the limitation of Android Apps.

4.3 Conclusion

We can only hope that the next versions of Android have overcome the actual limitations and that the future possibilities became a reality.