International Journal of Innovative Technology & Adaptive Management (IJITAM)
ISSN: 2347-3622, Volume-1, Issue-8, May, 2014

An Overview on Android Internal Systems and Internal Process communications

Rashmi Kumari, B-Tech 3rd year, Computer Science and Engineering, reshu.rashmikumari@gmail.com, C.V.Raman College of Engineering, Bhubaneshwar, Odisha.

Akash Kumar Athghara, B-Tech 3rd year, Electronics and Telecommunication Engineering,akash.athghara@gmail.com,Veer Surendra Sai University of Technology,Burla,Odisha.

Amit Kumar Gupta, B-Tech 3rd year, Computer Science and Engineering,amy.amitgupta@gmail.com,C.V.Raman College of Engineering,Bhubaneshwar,Odisha.

Dipankar Pramanik, Assistant professor, CSE, AIET, prdipu@gmail.com,Bhubaneswar, ORRISA.

Abstract.

Android is a linux based operating system which uses linux kernel. In this paper we will see how the boot up process of android is different from linux and how the different applications in an Android System communicate with each other. As an Android Application is made up of different activities so through this paper we will also come to know about how the activity changes their states and the whole lifecycle of an activity.

Keywords: *- Android Boot up, Activity Lifecycle, Zygote, Dalvik VM.*

1. INTRODUCTION

Android [1] is a free and open source operating system. It was initially developed by Android inc. but later in 2005 it was purchased by Google. In 2007 a group of 78 companies formed a group called Open Handset Alliance (OHA) to develop and distribute Android [4]. It is an operating system for low powered devices, those runs on battery and is full of hardware. Android applications uses hardware features through abstraction and provide a defined environment to run applications.
A Gartner forecast [8] stated in late 2010, that Android will "... challenge symbian for no. 1 position by 2014".That means, that Android is an increasing factor in smartphone computing. An Android application is written in java and run in a virtual machine which is called Dalvik virtual machine which executes its own byte codes [1]. Android introduces new extensions and features to the Linux kernel. The Binder framework for interprocess communication represents such a feature.

2. Background

2.1. Multitasking, Processes and Threads

Multitasking is the ability to execute multiple instances of programs or processesat the same time. An *operating system* therefore creates for every binary executable a certain memory frame with its own stack, heap, data and sharedmapped libraries. It also assigns special internal management structures. This iscalled a *process*.
The operating system must provide fair proportioning, because only one processcan use the CPU at the same time. All processes must be interruptible. Theoperating system sends them to sleep or wakes them on their time slot. Thiswork is done by a *scheduler*, supplying each process with an optimal time slot.
A *thread* is a process without own address space in memory, it shares theaddress space with the parent process. Processes can have child threads, and athread must be assigned to a process.

2.2. Process Isolation

Due to security and safety reasons, one process must not manipulate the data ofanother process. For this purpose an operating system must integrate a conceptfor *process isolation*. In Linux, the *virtual memory mechanism* achieves that byassigning each process accesses to one linear and contiguous memory space. Thisvirtual memory space is mapped to physical memory by the operating system.
Each process has its own virtual memory space, so that a process cannot manipulate the memory space of another process. The memory access of a process islimited to its virtual memory. Only the operating system has access to physicaland therefore all memory.

2.3. User Space and Kernel Space

Processes run normally in an unprivileged operation mode that means they haveno access to physical memory or devices. This operation mode is called in Linux*user space*. More abstractly, the concept of security boundaries of an operatingsystem introduces the term *ring*. Note, that this must be a hardware supportedfeature of the platform. A certain group of rights is assigned to a ring. Intelhardware [9] supports four rings, but only two rings are used by Linux. Theseare ring 0 with full rights and ring 3 with least rights. Ring 1 and 2 are unused.System processes run

in ring 0 and user processes in ring 3. If a process needshigher privileges, it must perform a transition from ring 3 to ring 0. The transitionpasses a gateway, that performs security checks on arguments. This transition iscalled *system call* and produces a certain amount of calculating overhead.

2.4. Interprocess Communication in Linux

If one process exchanges data with another process, it is called *interprocess communication* (IPC). Linux offers a variety of mechanisms for IPC. These are thefollowing listed: [10]

Signals: Oldest IPC method. A process can send signals to processes with thesame uid and gid or in the same process group.

Pipes: Pipes are unidirectional bytestreams that connect the standard outputfrom one process with the standard input of another process.

Sockets:A socket is an endpoint of bidirectional communication. Two processescan communicate with bytestreams by opening the same socket.

Message queues: Processes can write a message to a message queue that is readable for other Processes.

Semaphores: A semaphore is a shared variable that can be read and written bymany processes.

Shared Memory: A location in system memory mapped into virtual addressspaces of two processes, that each process can fully access.

2. Android Components

Android was developed by the Open Handset Allianceand Google and is available since 2008. [1] The basic components required for an Android are discuss below:

3.1. Kernel

Android is based on a Linux 2.6 standard kernel but enhanced with new extensions for mobile needs. These are kernel modules *Alarm*, *Ashmem*, *Binder*, powermanagement, *Low Memory Killer*, a kernel debugger and a logger. We will analyze the Binder driver in this work, that offers a new IPC mechanism to Linux.[7].

3.2. Programming Languages

Four programming languages are used for system development: Assembler, C,C++ and Java. The kernel has a small amount of Assembler but is mainlywritten in C. Some native applications and libraries are written in C++. Allother applications, especially custom apps, are written in Java. [4].

3.3. Java Native Interface

A distinction is made between programs compiled for the virtual machine andprograms compiled to run on a speci_c computation platform, like Intel x86 orARM. Programs compiled for a speci_c platform are called native. Because Javais executed in a virtual machine with its own byte-code, no native code can beexecuted directly. Due to the need to access low-level os mechanism like kernelcalls, Java has to overcome this obstacle. This is done by the *Java native interface*(JNI) [22], which allows Java to execute compiled code from libraries written inother languages, e.g. C++. This is a trade-off between gaining capabilities ofaccessing the system and decreasing the level of security in Java.

3.4. Dalvik Virtual Machine

The *Dalvik virtual machine* (DVM) [2] [5] runs the Java programmed apps. TheDVM does not claim to be a Java virtual machine (JVM) due to license reasons, but fulfills the same purpose. Java 5 programs can run in that environment.

The Sun JVM is stack based, because a stack machine can be run on everyhardware. Hardware and platform independence were major design principlesof Java. The DVM is register based for performance reasons and well adaptedto ARM hardware. This is a di_erent design principle, taking the advantage ofhardware independence for high performance and less power consumption, whichis essential for mobile purposes with limited battery capability. The possibility touse the Java native interface weakens the security guarantying property of Javato implicit checking the bounds of variables and to encapsulate system calls andthe force to use JVM de_ned interfaces to the system. The use of native librariescan allow bypassing the type and border checking of the virtual machine andopens the door to stack-overflow attacks. [4]

Even it is a security issue, the JNI is essential for the interprocess communication mechanism because the middleware of Binder are C++ libraries and mustbe accessed with JNI.

3.5. Zygote

Due to performance reasons, the DVM is started only once. Each new instanceof it is cloned. This is done by a system service called *Zygote*. [2]

First, it preinitializes and preloads common Android classes in its heap. [5]Then, it listens on a socket for commands to start a new Android application. Onreceiving a start command, it forks a new process with the loaded application.

This process becomes the started application and shares the heap with the originalZygote process by copy-on-write mapping and so the memory pages of Zygote'sheap are linked to this new process. While the application reads only from theheap, it stays shared. But when the application performs write operations on its heap, the corresponding memory page is copied and the link is changed to thenew page. Now the heap can be manipulated, without manipulating the originaldata from the parent Zygote process.

When an Android application forks, it uses Zygote's memory layout and therefore the layout is the same for each application.

3.6. Application Concept

Each Android application is composed from up to 4 different components. [12]
Each component has a special subject. Figure 3.1 presents the components as ahierarchically class diagram since they are actually Java classes.

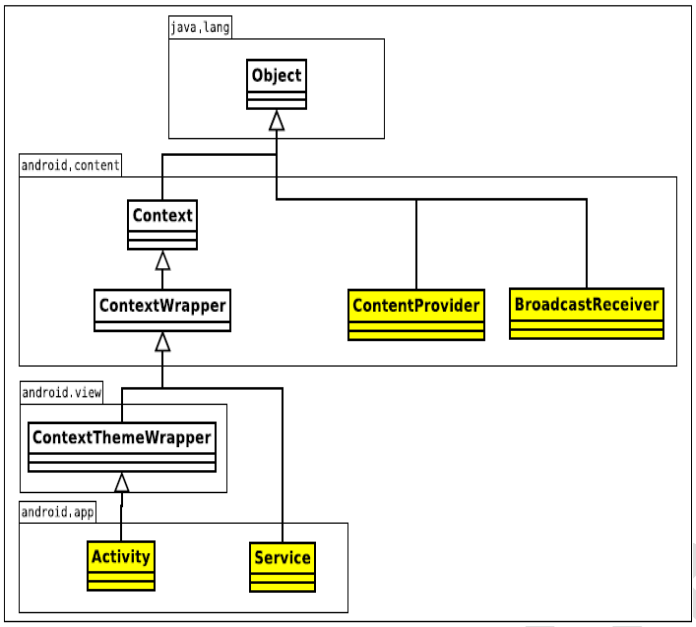

Figure 1: Application Components System

The *activity* represents the user interface of an application. It is responsiblefor performing the screen and receiving interaction created by the user. It is notintended to hold persistent data because it can be sent to sleep by the operatingsystem if another activity is brought to the front.
For long duration purposes Android offers the *service* component. All tasksrunning in the background of an application must be implemented here, becausea foreground service is only stopped if the system runs out of memory and appsmust be terminated to free memory..
The *broadcast receiver* is for receiving system wide messages, i.e. the messagethat a new SMS has come in is provided to all subscribers. A low battery levelwarning is also sent on this channel. *Broadcast reveivers*handle these messagesand marshal certain action, e.g. saving the state of an app in prospect to a soonshutdown of the mobile device.
The *application manifest* [8] keeps the information for Android about the component. In this file, the basic application con_guration is set. E.g., if anservice starts in its own process or if it is attached to local process. Listing 1gives an example of an Android application manifest XML file.

```
1 <?xml version=" 1 . 0 " encoding=" ut f□8"?>
2  <mani fest xmlns : android=" http : / / schemas .
android .com/apk/ r e s / android "
3     package="com. msi .manning .binde r ">
4     <application android :i con="@drawable/ i c on ">
5     <activity android : name=" . Act ivityExample "
android :label="@string /app name">
6 <intent□filter >
7 <action android : name=" android . i n t e n t . a c t i on
.MAIN" />
8 <category android : name=" android . i n t e n t .
category LAUNCHER" />
9 </intent□filter >
10 </ activity >
11 <service android : name=" . SimpleMathService "
12 android : process=" : remote ">
13 <intent□filter android : p r i o r i t y="25">
14 <action android : name="com. msi . manning .binder
ISimpleMathService "></ac t ion>
15 </intent□filter >
16 </service >
17 </application>
18 </manifes t>
```

Listing 1: Example Manifest

3.7. Component Communication Concepts

As different components have to exchange data, this is realized through intercomponent communication, or interprocess communication, if the specific component belongs to different processes (apps).
The communication works with so called intents. These are representations for operations to be performed. An intent is basically a datastructure which containsa URI and an action. The URI uniquely identifies an application component andthe action identifies the operation to be executed.
A service can be started, stopped and bound by IPC. Also the call and returnmethods are implemented by IPC.
A content provider can be queried by an activity via IPC and returns theresult accordingly. The Android source code files show an extensive use of IPCto exchange the abstract data.

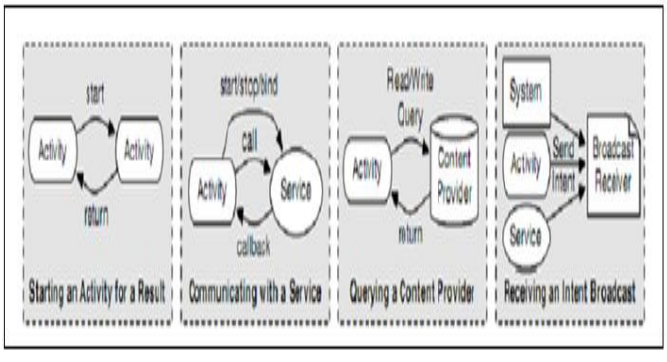

Figure 2: Application Components System

At this point, the importance of the IPC mechanism becomes apparent. TheAndroid OS with its framework is a distributed system and the major and keytechnology to achieve that design is the IPC Binder mechanism.

3.8. Security Concept

The security mechanism in Android consists of three layers. The basic layerconsists of a division of the persistent memory in two partitions, called systemand data. The system partition is mounted as read only to prevent systemdata manipulation. The data partition is the place where application states andpersistent data can be stored. Note, that the system partition can be remountedin write mode by the *App Store* application to install new apps.

4. ANDROID BOOTUP PROCESS

4.1. Power On

Master boot record (MBR) is a boot sector which contains partition table which has the information about how the device is partitioned in a structure. There is no MBR or partition when the device is started for the first time. When the phone is switched on, CPU will be in a no initialization state. Internal RAM is available and no internal clocks are set up. The device starts executing code located in the ROM and finds a specific block which has first Stage boot loader. The first boot loader points to a second stage boot loader, which is located in a known block. This “pointing” process is called raw partition table [6].

4.2. Boot loader

Boot loader is a code which is executed before android operating system runs. It loads kernel to the RAM and sets up the initial memories. Manufacturers use existing boot loaders or they create their own boot loaders.

- The First stage boot loader will find and setup the external RAM.
- Now Main boot loader is loaded and placed in external RAM as the RAM is available.
- The First important program is in the second boot loader stage which contains code for file systems, additional memory and network support etc.
- When the boot loader is done it goes to the linux kernel [10].

4.3. Linux kernel

A kernel acts as a bridge between hardware and software. It setups cache protected memory, scheduling and loads drivers. After initializing Memory management units and caches, virtual memory can be used and user space processes can be launched by the system. After finishing the setup Kernel looks for init process which can found under system/core/init and launch it [1].

4.4. Init process

This process is the root process. Every process will be launched from this process. Init process mounts directories like /sys, /dev, /proc. It will run init.rc script and system service processes. This script is located in system/core/rootdir in the Android open source project and describes system services, file system and other parameters [9].

4.5. Zygote

After starting various daemons like Android Debug Bridge (adbd), Radio Interface Layer Daemon (rild), etc, Init process initiates a process called Zygote. In java there is a separate instance of a Virtual Machine for each application. In android Dalvik, virtual machine is used as VM. So there is high consumption of memory and time because of different instances of dalvik VM for every application. Now Zygotes comes into play. It enables shared code across Dalvik VM, lower memory footprint and minimal startup time. Zygote process starts at system boot up and it preloads and initializes core library classes. After initialization, zygote process waits for socket request coming from the runtime process. If any request comes then it forks starts processes with VM instances [6].

4.6. Runtime process

The next init initiates the Runtime process and this process starts the service manager. All the services should be registered with the service manager and it provides local lookup service and binds services given their name. The runtime requests zygote to start system server process. Zygote splits and starts up a new dalvikvm instance and starts the service. To control display device and audio output device the system server starts surface flinger and audio flinger. These services get registered with the service manager so that other applications can use display and audio. Now the system server will start all the core platform services and hardware services like activity manager, window manager and power manager etc. All of these services will get registered with the service manager [2].

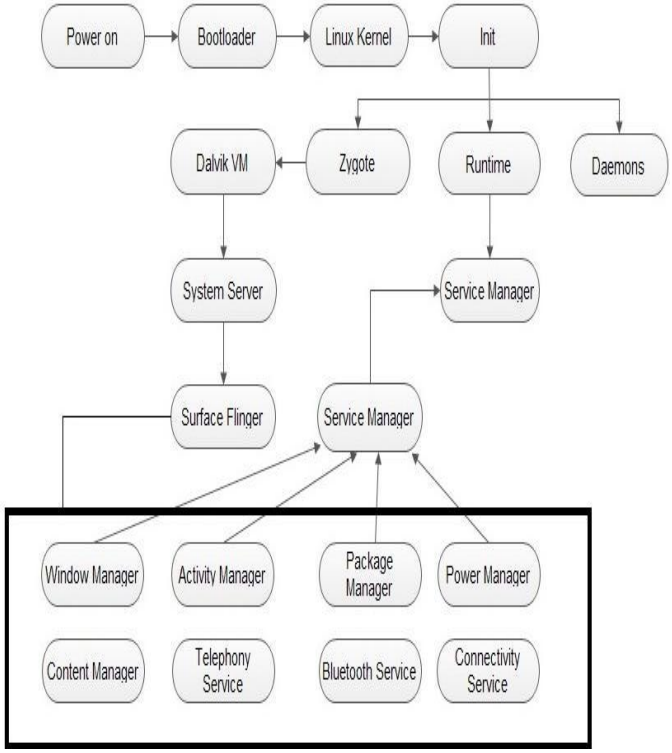

Fig 3: Boot up Process

5. INTERPROCESS COMMUNICATION (IPC)

At this point home screen or idle screen is launched. Activity manager will send a request to zygote to initiate the “home” activity and in return Zygote will fork a new process a dalvik VM and home activity. Now for each application launched by the user, Zygote will fork each time and create a new dalvik VM instance inside a new process. A unique user id is assigned to each application. An application has access to only those files which it needs and these permissions are set up by the system [3]. Applications run in separate processes, so to communicate with each other and with the system services an IPC (Inter Process Communication) is needed and this mechanism in android is known as Binder and is based on shared memory. On registration of each process with the service manager, it gets a reference called a context object. Let there be an application A and service B which is running in spate processes and wants to communicate with each other application. A passes the name of the service to context and requests for service. B in return context sends a reference to the service to A [3]. After getting the reference app, A calls a method which is intercepted by the Binder which arranges the object and passes the reference to Receiver. The object is serialized because a proxy object is passed and not the original objects. There is a thread pool maintained by the binder at Receiver B side and one of the threads receives the incoming call, locates the actual object and makes the call. Return value is passed back to the application A [6].

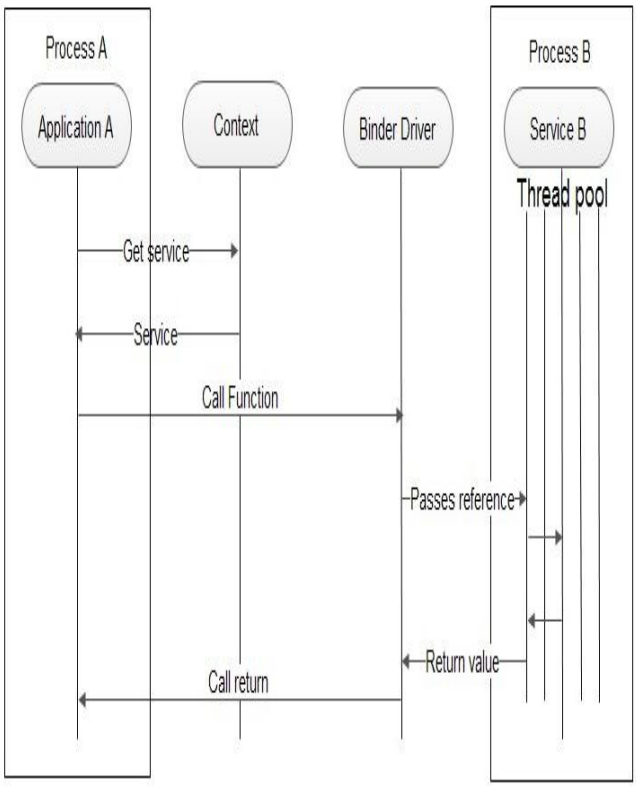

Fig 4: Interposes Communication

6. APPLICATIONS

An android application consists of four components:

1. Activity: An activity is a single screen of an application with which user can interact like click a photo, dial a number etc. Intents are used for transition between different activities and each application can have multiple activities.
2. Service: A service is a component that does not provide any user interface but runs in the background to perform long-running tasks.
3. Content Provider: To exchange data between different applications, a component is used. It is called content provider which handles retrieval of data and stores data in database files or on a network.
4. Broadcast Receiver: The broadcast messages from other application or from system are called intents and the component which responds to these intents is called broadcast receiver [2].

6.1. Activity States

Activities can be seen as website. Just as a website contains multiple web pages, an android app contains multiple activities. One webpage can redirect to another page and so on. In an android app one activity can redirect to another and so on. All the handling activities is done by activity Manager. Its task is to create, destroy and manage activities.

6.1.1. Activity states

1. **Active/Running:** An activity is said to be in running state if it is completely visible and the user can interact with it. There can be only one running activity at a given time.
2. **Paused State:** An activity is said to be in paused state if it is not in focus but partially visible. For example while using an app on android if any notification or a dialog box appears then the activity of the app goes in paused state. While in paused state, an activity still maintains all states. It remains attached to the window manager and it can be killed by OS under low memory.
3. **Stopped:** An activity is said to be in 'stopped state' when it is not at all visible on screen but it is still alive and maintains all states. To fulfil the resource requirements of higher priority activities, it can be killed by OS.
4. **Destroyed state:** An activity is said to be destroyed when it is no longer in memory. OS destroys an activity after a 'paused' or 'stopped state' to free the resources [7].

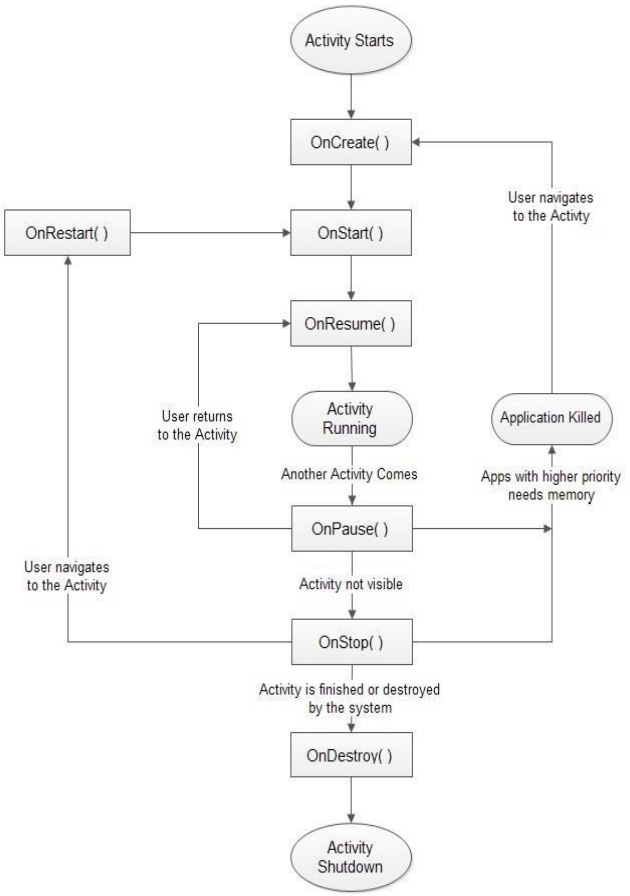

Fig 5: Activity Lifecycle 4.1.2. Activity Lifecycle Methods

1. **Oncreate( ) :** When an Activity is launched the first method called is the oncreate( ) method All the User Interface creation and initialization of data elements is done in this method. A bundle object as parameter is passed in this method to restore the UI state [7].
2. **Onstart( ):** Just before an Activity is being visible to the user, Onstart() method is called and if there is any task needed to be performed just before the Activity becomes visible to the user, it is defined in this method as ramping up frame rates [5].
3. **OnResume( ):** Now the activity is in 'active' or 'running state' and user can interact with the activity. This method can also be called when the activity is in 'paused state' to make it in 'running state' [8].
4. **Onpause( ) :** When an activity is in running state, a user might navigate to other parts of the system or the system is about to put an activity in background then Onpause( ) method is called. After this an activity can be killed by the system or Onresume( ) method can be called to resume an activity [5].
5. **Onstop(**): When an Activity is not visible to the user, Onstop method is called and the application can be killed at anytime by the system in case of low memory. After this, an activity is either restarted or destroyed [8].
6. **Onrestart( ):** In 'stopped state' an activity can't be resumed but it can be restarted by calling Onrestart method and it is always followed by onstart method [5].
7. **Ondestroy( )** : This method is called just before an activity is destroyed because the activity has finished or system kills it to save space. This is the last method called on an Activity [5].

7. CONCLUSION

Android has emerged as a strong competitor in mobile sector as it is supported by large companies especially by Google. Manufacturers can modify the system as per their needs due to Android's openness and extensibility. Today Android operating system is not only used in Mobiles and tablets, its implementation in electronic devices is increasing rapidly. Smart TV and Smart Camera are examples of new implementation and in future android will be in many household devices like washing machine, Oven and many more.

8. REFERENCES

[1] Openhandset Alliance. Android overview, 08 2011.URLhttp://www.openhandsetalliance.com/android_ overview.html.

[2] Bornstein. Dalvikvm internals, 2008 google i/o session, 01 2008.http://sites.google.com/ Site/io/dalvik-vm-internals.

[3] Stefan Brahler," Analysis of android architecture", Department of computer science, Karlsruher institute of technologies, Germany, june 2010

[4] http://www.developer.android.com, 23 Oct, 2012.

[5] David Ehringer. Dalvik virtual machine, 03 2011. URLhttp://davidehringer.com/software/android/The_Dalvik_Virtual_Machine.pdf.

[6]Goolge. Android documentation - what is android, 08 2011.URLhttp://developer.android.com/guide/basics /what-is-android.html.

[7] Chin Felt Greenwood Wagner. Analyzing inter-application communication in android, 06 2001.

[8] Google. The android mainifest xml _le, 08 2011.URL http://developer.android.com/guide/topics/manifest/manifest-intro.html.

[9] Intel. *Intel 64 and IA-32 Architectures Software Developer's Manual*. Intel,2011.

[10] David A Rusling. *The Linux Kernel*. 1999.