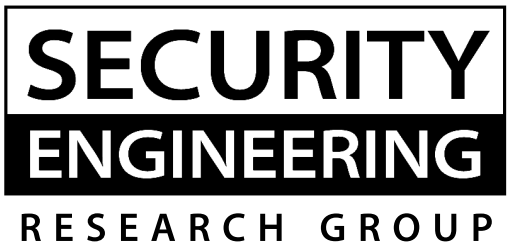
Technical Report

Analysis of Dalvik Virtual Machine and Class Path Library



Constrained Intents: Extending Android Security for Intent Policies (EASIP)

Security Engineering Research Group,
Institute of Management SciencesPeshawar, Pakistan
http://serg.imsciences.edu.pk

November, 2009

Disclaimer

Contributors

Sohail Khan (sohail.khan@imsciences.edu.pk)
Shahryar Khan (engrshahrs@gmail.com)
Syed Hammad Khalid Banuri (hammadbanuri@gmail.com)

Mohammad Nauman (nauman@imsciences.edu.pk)
Masoom Alam (mmalam@imsciences.edu.pk)

Acknowledgements

The preparation of this technical report has been supported by Grant No. ICTRDF/TR&D/2009/03 from the National ICT R&D Fund, Pakistan to Security Engineering Research Group, Institute of Management Sciences.

Table of Contents

List of Tables

List of Figures

Glossary

VM	Virtual Machine
SDK	Software Development Kit
OHA	OPen Handset Alliance
AdMob	Mobile Advertising Platform
UID	User ID
ARM	Advanced RISC MAtchine
J2ME	Java 2 Micro Edition
CLDC	Connected Limited Device Configuration
API	Application Program Interface
BSD	Berkley Software Distribution
SSL	Secure Socket Layer
ICU	International Components for Unicode
JNI	Java Native Interface
GPL	Generic Public License
SHA	Secure Hash Algorithim
NOJIT	Not-Just-In-Time
CRC	Cyclic Redundancy Check
JVM	Java Virtual MAchine
Re-Tools	Reverse Engineering Tools

Abstract

Analysis of Dalvik Virtual Machine and Class Path Library

Security Engineering Research Group,
Institute of Management Sciences
http://serg.imsciences.edu.pk

November, 2009

Android is an open source complete software stack (that includes operating system, Linux kernel, middleware and some useful applications) for mobile platforms; proposed by Open Handset Alliance, a global alliance of leading technology and mobile industries. Android platform has its own virtual machine dalvik. Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple instances of the VM efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included dx tool. The Dalvik VM relies on the Linux kernel for underlying functionalities such as threading and low-level memory management. In this report we have discussed virtual machines, specifically Dalvik virtual machine in detail and their runtime libraries in comparison with JVM's libraries. The report answers questions, such as why Dalvik was needed and how it overcomes the deficiencies of its counterpart Java virtual machine.

Moreover, this report also gathers some statistics related to the Android and iPhone market place. Most of comparative statistics suggest that Android is still far behind iPhone in the consumer market, both in terms of revenue generation and size captured. However, it is also evident how quickly Android has advanced in market to its current size of 7000 applications and almost 3000 publishers. Android new projects increased to almost doubled the number in just one month i.e., 325 projects in September to 625 in October.

Chapter 1

Introduction

Today everyone is on the move, and so the usage of mobile devices has increased dramatically in last few years. Today's smart phones have enhanced the capabilities of traditional cell phones to great deal. They are provided with such useful features like high resolution displays, increased processing power, greater storage capacity, text and multimedia messaging, multimedia audio and video players, audio and video call conferencing, GPS service, enhanced local wireless communication (Infrared to bluetooth, and now from blue tooth to Wi-Fi) and so on. Along with some general and professional communities, these later features have certainly grasped the attention of business community for the usage of smart phones for their precious business centric applications.

The rise in the usage of mobile devices (cell phones, PDAs, and smart phones) and rapid growth in demand of their software application development has convinced the manufacturers towards open source software stacks. Open source shift the control of software from vendors to customers. The source code of open software is available to everyone; hence, users or third party organizations can learn from it and can extend it according to their needs. Open source leads to rapid growth and development in software industry. It also allows a common platform for different vendors to develop and enhance the capabilities of software distributions. Experts (belonging to different companies) collaborate with each other and the new ideas travel the globe in an instant. An example of such new and emerging opensource platform is the Google's Android [1] smartphone.

1.1 Android

Open Handset Alliance (OHA) [19], a global alliance of leading mobile and technological advancement industries come up with open source mobile platform – Android. Android is complete software stack for mobile devices, including a new operating system, Linux kernel, middleware and having all those modern applications, which are available in any of modern mobile handsets (developed by mobile industry's giants like Nokia, Samsung, Sony Ericsson, etc).

Android also includes a set of core libraries that provide most of the functionality available in the core libraries of the Java programming language. Every Android application runs in its own process, with its own instance of the Dalvik [2] virtual machine. Dalvik has been written so that a device can run multiple instances of this VM efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included dx tool. The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management. Further details of dalvik are in upcoming chapters.

Android SDK is available for software developer to produce some powerful and innovative mobile applications to provide the mobile clients with better experience. Android provides such a flexible reusable architecture, which allows the new application to incorporate the features of any of previously built android's functionalities (applications and services).

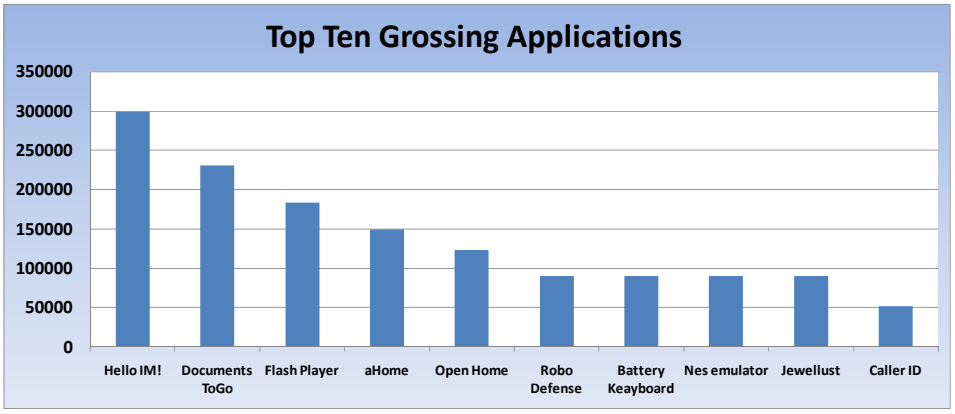


Figure 1.1: Top 10 Grossing Applications in the US Android Market based on Estimated Downloads and Current Price.

Android is provided with so much attractive features and technological world's experts are getting involved in developing new android applications. The graph presented in Fig 1.1 mentions the top 10 grossing applications in the US Android Market based on estimated downloads and current price [8]. The top application (named Hello IM [15]) is an Android AOL instant messenger. Hello IM, is provided with almost all those features that technologically advanced instant messenger could have. Like, you can stay with your buddies through exchange of text messaging, emotions pictures, can customize ringtone for individual buddies; you can change the notification by applying different colors of LCD or through vibration; and so on. The top application is then followed by some system utility applications and some games.

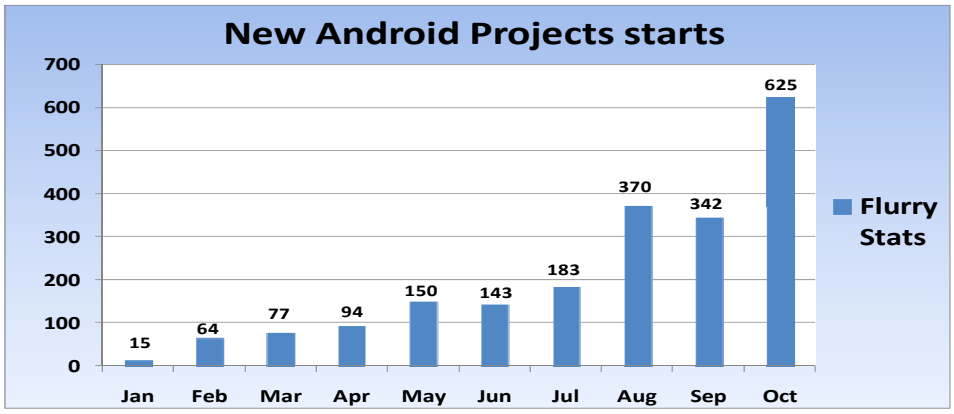


Figure 1.2: Monthly Statistics of new Android Projects.

The technological world is bustling with the news of latest Android developments, from the upcoming Google's Maps with navigation application to upcoming Motorola Droid [14], which has already drawn attention of consumers in recent weeks. The Motrolla Droid will be the first device, which would be able to execute the Android 2.0. The development within Android community is growing at rapid speeds. Mobile analytics firm Flurry released stats recently showing just how much developers are jumping on the Android bandwagon [11].

The Flurry statistics in Figure 1.2 shows that in just one month (i.e. from September to October) the number of Android projects started has almost doubled. The boom began with the launch of the MyTouch, Hero and Cliq devices, and is getting even stronger with the upcoming launch of the Droid from Motorola.

According to stats of AdMob [3] (Mobile Advertising Platform), the revenue generated by Apple's iPhone store through selling their applications, is worth $200 million per month (which reaches about $2.4 billion per year). Whereas, Android App

produces monthly revenue of about $5 million ($60 million worth yearly). Android has a long way to go to get anywhere near the size of Apple's Application Store both in terms of revenue and size. Apple hit 65,000 applications in July compared to 7,000 applications in the Android Market today [7]. However, Android Market is growing faster to its current size of 7000 applications and almost 3000 publishers.

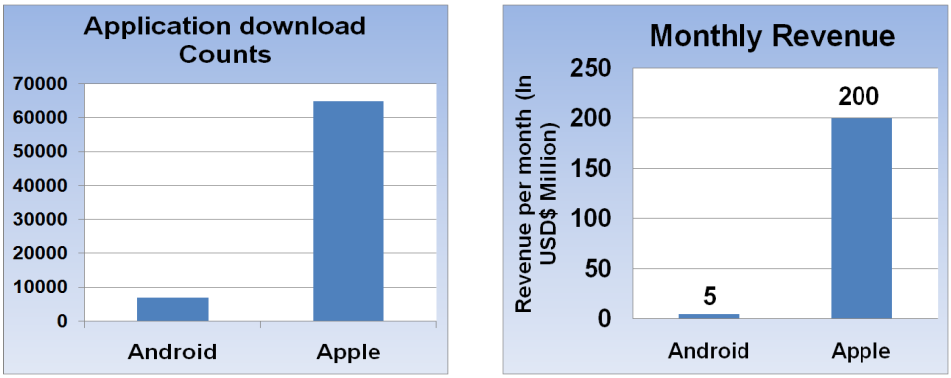


Figure 1.3: Application Downloads Count and Monthly Revenue Comparison for Android and iPhone.

If you closely observe the above graphs, you may find that first graph is not matching the statistics of second graph or vice versa. The application download counts for Apple is about 9.26 times greater than that of Android application download counts. Whereas, the monthly revenue of Apple is exactly 40 time that of monthly revenue generated by Android. The difference between both graph's statistics is mainly because Android users are able to try out an application for 24 to 48 hours without their credit card being charged. Therefore actual sale of an application is somewhat lower than the addressed download counts [7]. Besides that we believe that the difference in cost of per application download also creates discrepancy between information of both Apple and Android.

AdMob team surveyed almost 1000 users of Android & iPhone smart phone and come up with some interesting stats about the application downloads. According to their Mobile Metric Report [4], application average downloads per month for Android, & iPhone devices are approximately 9 & 10 respectively. Android monthly average paid download is one, where as the paid downloads of iPhone are more than twice that of Android. Rest of the downloaded applications were free to download. Another statistics in Figure 1.4, declared by same report, shows that how much percent of the total users (of Android & iPhone) downloads one or more paid applications per month. The report also mentions that more than 90 percent of Android and iPhone OS users browse and search for applications directly on their mobile device instead

of their computer. The report concludes that the main reason of deriving users to purchase paid application is their upgradation from light version.

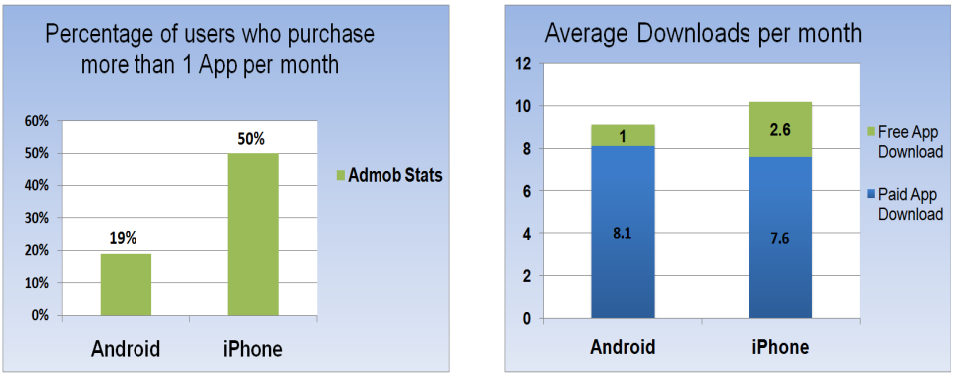


Figure 1.4: Average Applications Downloads Comparison for Android and iPhone.

Most of the above discussed stats show that there is a long way to go for Android in order to beat iPhone in terms of market share. However, some of the stats also mentions that how quickly Android is advancing in current market and new projects of Android starts has almost been doubled in just one moth (from September to October this year). Its author's contention that in near future the Android will be one of the leading giants of technological world.

Chapter 2

Dalvik Virtual Machine

A Virtual Machine (VM) is a software environment that can be an emulator, an operating system or a complete hardware virtualization, that has an implementation of resources without the actual hardware being present.

As an emulator it allows applications and operating systems to run on hardware that has a different processor architecture than the present one, while as an operating system VM virtualizes the server on operating system, and in-case of hardware virtualization two or more different operating systems can run simultaneously on the same hardware. Thus the main idea of virtual machine is to provide an environment that can execute instructions other than those associated with the host environment regardless of the hardware and software.

2.1 Types of Virtual Machines

VMs act as a hardware for performing operations just as if real hardware is present. VM can be divided into two categories based on its working and functionality:

1. System Virtual Machine (supports execution of a complete operating system)
2. Process Virtual Machine (supports execution of a single process)

Based on its architecture VMs are divided into two categorizes.

1. Stack based VM (uses instructions to load in a stack for execution)
2. Register based VM (uses instructions to be encoded in source and destination registers)

Below we discuss a brief comparison between the two broad catogries.

System VM vs Process VM

Virtual Machines mainly divide into two broad categories, i.e. System VM (also known as hardware virtual machine) and Process VM (also known as application virtual machine) [23]. The categorization is based on their usage and level of correspondence to the associated physical machine. The system VM simulates the complete system hardware stack and supports the execution of complete operating system. On the other hand, Process VM adds up layer over an operating system which is use to simulate the programming environment for the execution of individual process. Virtual Machines are use to share & specify appropriate system resources to the software (could be multiple operating systems or an application); and these softwares are limited to their given resources provided by VM. Virtual Machine Monitor (also known as Hypervisor) is the actual software layer which provides the virtualization. Hypervisors are of two types [9], depending on their association with the underlying hardware. The hypervisor that takes direct control of underlying hardware is known as native or bare-metal VM; while hosted VM is distinct software layer that runs with in operating system and hence have indirect association with the underlying hardware. The system VM abstracts Instruction Set Architecture (ISA)[1], which is bit different from that of real hardware platform. The main advantages of system VM includes consolidation (it allows the multiple operating systems co-existence on single computer system with strong isolation to each other), application provisioning, maintenance, high availability & disaster recovery. Besides later advantages, regarding development aspects their advantages are that it allows sandboxing, faster reboot and better debugging access [26].

The application or process VM allows normal execution of application within underlying operating system to support a single process. We can create multiple instances of process VM to allow the execution of multiple applications associated with multiple processes. The process VM is created when process begins and it ends when process gets terminated. The main purpose of process VM is to provide platform independency (in terms of programming environment), means allow execution of application in same manner on any of underlying hardware and software platforms. In contrast to system VM (where low-level abstraction of ISA is provided), process VM abstracts high level programming language. Process VM is implemented using an interpreter; however the comparable performance to the compiler based programming languages is achieved through just-in-time compilation method [24].

[1] Instruction Set is a set of all those tasks that a processing unit of a system can perform and in case of virtual machine it is set of all those instructions that an interpreter can execute.

Two of the most popular examples of process VMs are Java Virtual Machine (JVM) and Common Language Runtime; used to virtualize the Java programming language & .NET Framework programming environment respectively.

Stack Based VM vs Register Based VM

For many years, in virtual machine architecture, the stack based VMs [23] was the architectural choice due to simplicity of the VM implementation and the size of executables for stack architectures. Registered-based VM [23] can be an attractive alternative to stack-based VM due to the reason that it allows a number of executed VM instructions to be substantially reduced. A study on analysis of different VM shows that register-based architecture requires an average of 47% less executed VM instructions than the stack based. On the other hand the register code is 25% larger than the corresponding stack code but this increased cost of fetching more VM instructions due to larger code size involves only 1.07% extra real machine loads per VM instruction which is negligible. The overall performance of the register-based VM is that it take, on average, a 32.3% less time to execute standard benchmarks [22].

Due to the benefits stated above Google has chosen a register-based VM for the Android platform. Apart from the licensing issues with Sun, there are a number of technical advantages for selecting a registered-based VM [22]:

- A register VM is likely to have an intrinsic performance advantage over a stack VM when hosted on a pipelined processor.
- Byte code verification is likely to be faster on a register VM (i.e., faster startup times) because stack height integrity checks will be greatly simplified.
- A register VM will be more forgiving of incorrect code (in the VM, generated by the compiler, code corrupted during program transmission or storage attacked by malware) than a stack VM.

2.2 Dalvik Virtual Machine

Dalvik [2] is a virtual machine that is designed specifically for the Android platform. Named after the fishing village of Dalvik in Iceland, it was originally written by Dan Bornstein.

Unlike most of virtual machines that are stack based, Dalvik architecture is register based. It is optimized to use less space. The interpreter is simplified for faster

execution. It executes its own Dalvik byte code rather than Java byte code. Byte codes are discussed in the upcoming section.

2.2.1 Motivation behind Dalvik

All mobile systems features little RAM, low performance CPU, slow internal flash memory, and limited battery power. Therefore, a need was felt for a VM that could provide better performance with limited resources. So came Dalvik, designed to run on Linux kernel, which provides process threading, pre-processing for faster application execution, User ID based security procedures and inter-process communication. Dalvik works on low resourced ARM devices (Advanced RISC Machines), is 32-bit processor architecture based on Reduced instruction set computer developed by ARM limited). ARM processors are used because of their simple architecture making it suitable for low power devices such as cell phones. Dalvik can also be ported to run on x86 systems.

2.3 Attributes of Dalvik VM

2.3.1 Architecture

Dalvik is a register based architecture making it faster and performance efficient for running application code. It has to operate on Dalvik byte code rather than Java byte code. Register based VM has potential advantages over the stack based VM as discussed in the Section 2.1. Now we discuss the currently supported functions [12].

The supported functionalities are:

- Dalvik execution file format.
- Dalvik instruction set
- J2ME CLDC API
- Multi-threading.

The supported platforms are:

- Baseline systems are intended to be a flavor of UNIX

- Linux
- BSD
- Mac OSX

2.3.2 Runtime framework and supported libraries

Runtime framework is a layer where a programmer starts building their applications. It consists of a virtual machine that execute Dalvik byte code (that is built out of the Java byte code and a core API).

The cell phones of today don't have sufficient resources as that of a PC, so traditional Java (as requires more resources) can not be used for mobile systems, in general. It is therefore required that a set of core functionality libraries optimized for these limitied capabilities devices be deployed.

In case of Dalvik,the supported libraries includes:

1. dalvik/libcore (written in C/C++)
2. dalvik/vm/native (written in C/C++)
3. OpenSSL (for encryption)
4. zlib (free, general-purpose, data-compression library)
5. ICU (for character encoding)
6. java packages (including java.nio, java.lang, java.util)
7. Apache Harmony classlib (including Apache HttpClient)

Dalvik VM itself is written in portable C. However it has one non-portable component of the runtime called JNI call bridge.

2.3.3 License of Dalvik

Dalvik has licensed its source code under the Apache license [5], making it attractive and in-expensive for mobile phone carriers, since they can use and modify it without paying licensing fees. Moreover, the Apache license lets them close any part of the code they want. Comparative to Java machines that are either closed source making them expensive or open source but not "businesses like" (e.g., GPLv2) [16].

2.3.4 Security issues

Dalvik has an additional layer for security, including process separation and file permissions to the underlying Linux platform. Dalvik is relatively secure because it provides sharing of code between processes without giving it permissions to edit the shared code.

Chapter 3

Dalvik Internals

3.1 Byte-code

A byte-code is a form of instruction set that is designed for efficient execution by a VM. It is contained in a binary file that has an executable program. Byte-code is called "Byte" code because each of the opcode is one byte in length (however length of instruction code differs) [18]. A complete understanding of byte-code and operations performed on it, is important for debugging and doing performance and memory usage analysis.

3.1.1 Java Byte-code

Understanding of Java byte-code here is important because the Dalvik byte-code is converted from the Java Byte-code. So one must know about structure of Java byte-code in order to know Dalvik byte-code and its conversion method. Java byte-code is generated by `javac` compiler. The size and execution speed of the code depends on the byte code length, format and order. For details about the Java byte-code we refer the readers to [20].

3.1.2 Dalvik Byte-code

Dalvik operates on its own form of byte-code known as Dalvik byte code. This byte code is created from the Java byte code. Reason for using its own byte is obvious. It prepares its byte code for optimal performance before it is executed.

3.2 Dex file format

Dalvik Executable are complied by Dalvik VM, and are zipped into a single .apk (Android Package) file on the device. The .dex files can be created automatically by translating compiled applications written in the Java programming language.

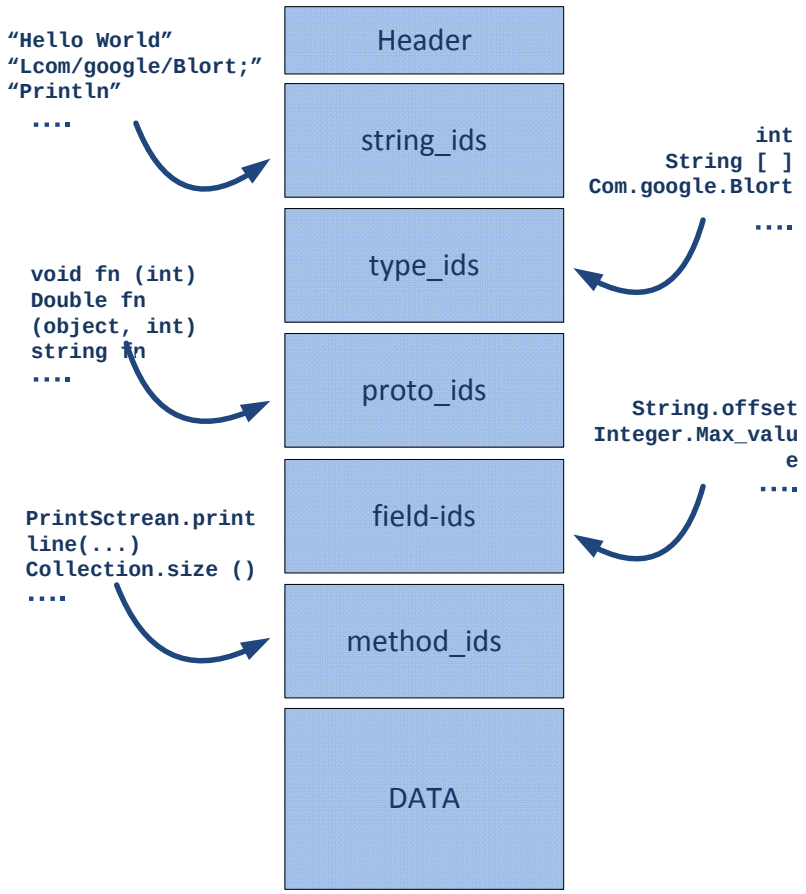


Figure 3.1: Dex File Anatomy [6].

The `.dex` file is divided into different constant pool sections. Each constant pool section is of a particular type. The "`string_ids`" section identifies all strings included in the `.dex` file. For example, an explicit coded string in the source code or the name of the method or function. At the bottom it contains a series of class definitions as a single `.dex` file may contain multiple classes.

Following are the structural formats for the tables and lists that are maintained. Every table and list contains their respective entries in its own format [2]:

File Header

Simple header is contained within the start of .dex file, with some checksums and offsets to other structures.

Offset	Size	Description
0x0	8	'Magic' value: "dex\n009 \0"
0x8	4	Checksum
0xC	20	SHA-1 Signature
0x20	4	Length of file in bytes
0x24	4	Length of header in bytes (currently always 0x5C)
0x28	8	Padding (reserved for future use?)
0x30	4	Number of strings in the string table
0x34	4	Absolute offset of the string table
0x38	4	Not sure. String related
0x3C	4	Number of classes in the class list
0x40	4	Absolute offset of the class list
0x44	4	Number of fields in the field table
0x48	4	Absolute offset of the field table
0x4C	4	Number of methods in the method table
0x50	4	Absolute offset of the method table
0x54	4	Number of class definitions in the class definition table
0x58	4	Absolute offset of the class definition table

Table 3.1: Performance Evaluation Results

String Table

The length and offsets for every string in the Dex file including string constants, class names, variable names and more are stored in this table. These entries have the following format:

Offset	Size	Description
0x0	4	Absolute offset of the string data
0x4	4	Length of the string (not including the null-terminator)
0xC	20	SHA-1 Signature
0x20	4	Length of file in bytes

Table 3.2: String Table

Class List

The .dex file contains classes or the reference to the classes used, and is maintained in class list. Each entry has the following format:

Offset	Size	Description
0x0	4	String index of the name of the class

Table 3.3: Class List

Field Table

All classes defined in the `.dex` file have their respective fields that are stored in this table. Format is as below:

Offset	Size	Description
0x0	4	Class index of the class this field belongs to
0x4	4	LString index of the field name
0x8	4	String index of the field type descriptor

Table 3.4: Field Table

Method Table

Dex file contains the methods of all classes that are being defined. Each method is stored in this table and has the following format:

Offset	Size	Description
0x0	4	Class index of the class this field belongs to
0x4	4	String index of the method name
0x8	4	String index of the method type descriptor

Table 3.5: Method Table

Method List

The methods defined in a particular class are listed in the method list, that starts with 32 bit integer having the number of items in the list followed by the entries.

Offset	Size	Description
0x0	4	Method Index
0x4	4	Access Flags
0x8	4	Throws list off
0xC	4	Absolute offset of header for code that implements method

Table 3.6: Method List

Class Definition Table

Every class has its definition or has a method or field accessed by code and is contained in this dex file. The format for each entry is as follows:

Offset	Size	Description
0x0	4	Class Index
0x4	4	Access Flags
0x8	4	Index of superclass
0xC	4	Absolute offset of interface list
0x10	4	Absolute offset of static field list
0x14	4	Absolute offset of instance field list
0x18	4	Absolute offset of direct method list
0x1C	4	Absolute offset of virtual method list

Table 3.7: Class Definition Table

Field List The pre-initialized fields in a class are stored in the field list. The format of each entry is such that the number of entries are followed by the entries themselves, which is 32-bit integer of which the list is formed.

Offset	Size	Description
0x0	8	Index of string or object constant or literal “primitive constant”

Table 3.8: Field List

Code Header

The information about implementation of a method is stored in a code header.

Offset	Size	Description
0x0	2	Number of registers used by this method
0x2	2	Number of inputs this method takes
0x4	2	Output size
0x6	2	Padding
0x8	4	String index of the source file name
0xC	4	Absolute offset of the actual code that implements this method
0x10	4	Absolute offset of the list of exceptions this method can throw
0x14	4	Absolute offset of the list of address and line number pairs for debugging purposes
0x1C	4	Absolute offset of the local variable list of this method (includes arguments to the method and "this")

Table 3.9: Code Header

Local Variable List

The local variables used in a method are stored in the local variable list. Its format is a 32-bit integer that contains the number of items in list.

Offset	Size	Description
0x0	4	Start
0x4	4	End
0x8	4	String index of variable name
0xC	4	String index of variable type descriptor
0x10	4	Register number this variable will be stored

Table 3.10: Local Variable List

3.3 Conversion of Java Byte Code to Dalvik Byte Code

Every Java program (as usual in Java) is compiled to byte code. The Java byte-code is then transformed into Dalvik byte-code with the help of "`dx`" tool and stored in `.dex` file, and that's on what Dalvik performs operations such as verification and optimization.

Developers write their own applications in Java and at build time the Java code is converted to a format compatible with Dalvik VM. The Android SDK has a specialized tool called "`dx`" tool which converts the `.class` files of Java to the Dalvik executable files. The `dx` tool arranges all the Java class files and eliminate all the redundant information that is present in the class files. The initial file loading and parsing procedures that happens again and again are eliminated. Finally, the byte code from the original class files is written into the new form which is executed by the Dalvik VM.

In general, a Java `.class` file contains a number of different method signatures used in the code. These signatures are duplicated if referenced in different Java classes. In other words, each of these classes references to the methods that takes the same

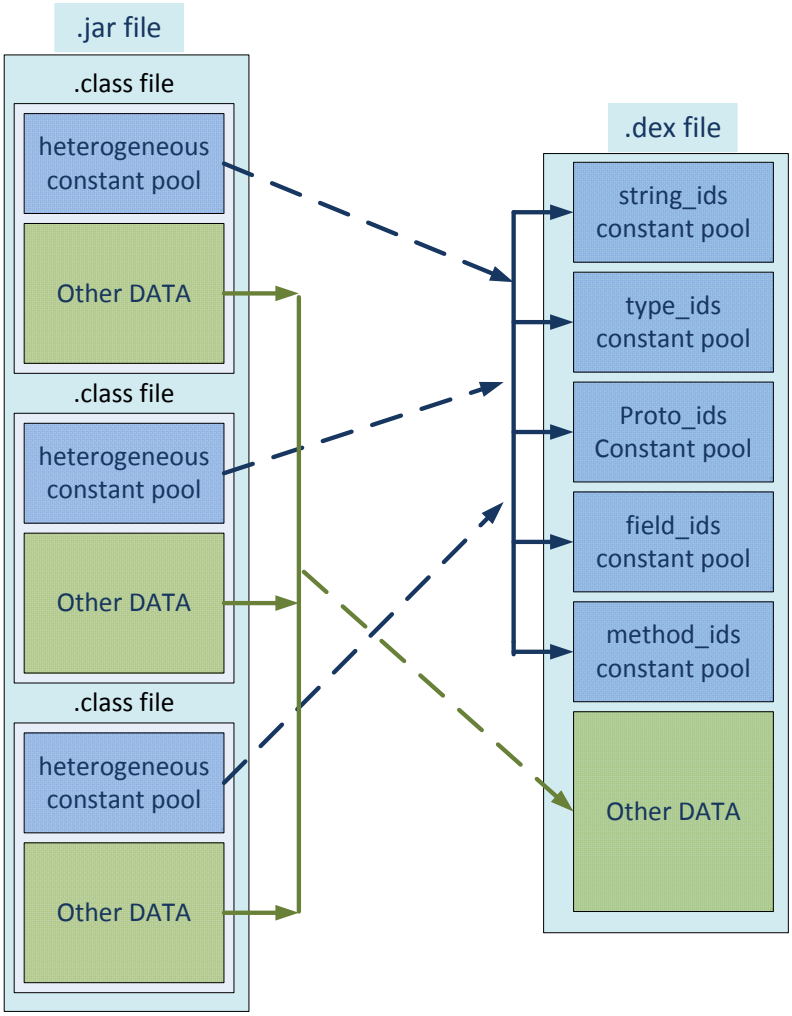


Figure 3.2: Jar to Dex Conversion [6].

arguments and the same return type. So, each one of these classes will have to include the same method signatures and hence the efforts of parsing the files are duplicated. In addition to this, there is a large number of strings included that labels the internal bits of the class files.

On the other hand, when the Java class files are converted to the Dalvik dex files, only a single dex file is created and all of the class files are included in this file. In the dex conversion all of the separate constant pools are collapsed into a single shared constant pool. This way not only is redundant information eliminated but storage space to store the shared constant pool is also conserved.

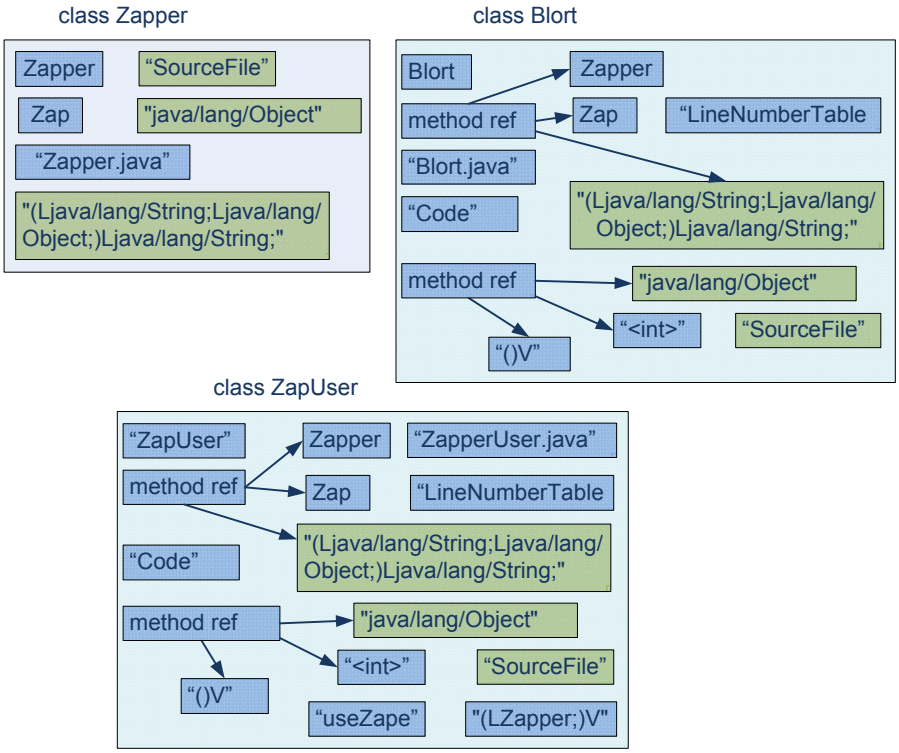


Figure 3.3: An Example of three different Java Classes [6].

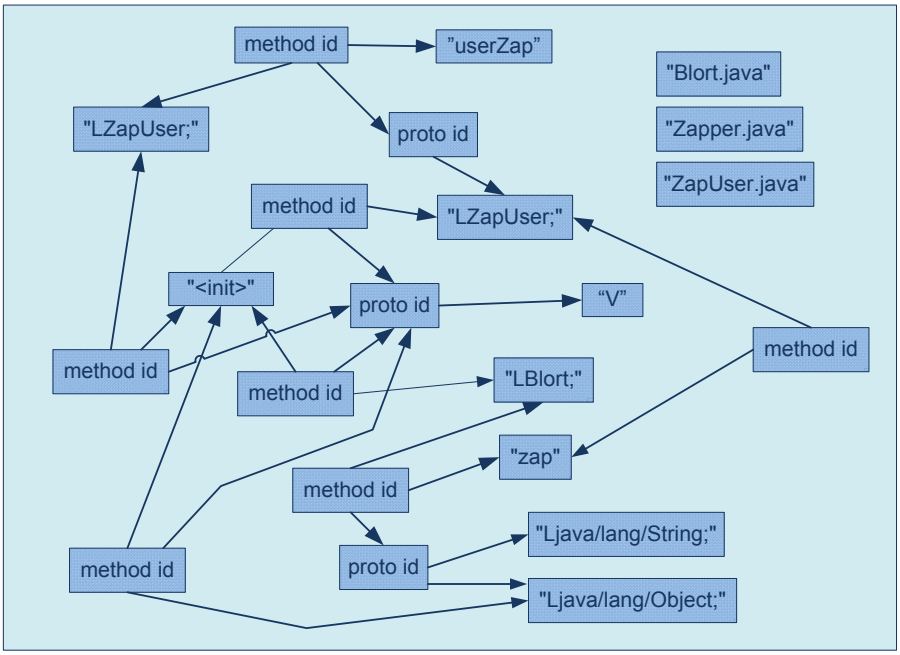


Figure 3.4: The resultant Dex file from the Java Classes [6].

3.3.1 Size Comparison of Dex & Jar Files

From this table it is evident that even the uncompressed dex file is taking less space than the compressed jar file.

Contents	Uncompressed jar File		Compressed jar files		Uncompressed dex file	
	In Bytes	In %	In Bytes	In %	In Bytes	In %
Common System Libraries	21445320	100	10662048	50	10311972	48
Web browser Application	470312	100	232065	49	209248	44
Alarm Check Application	119200	100	61658	52	53020	44

Table 3.11: Size Comparison between Jar & Dex files [6].

Chapter 4

Working & Optimization of Dalvik VM

4.1 Capabilities of a typical VM & Issues of Interest

Following are some issues described in system resources perspective, that a VM should be capable of:

4.1.1 System Memory & Overhed

The minimum size of the memory for the virtual machine should be set based on the recommendations of the operating system. So sharing of byte-code (also known as class-data) is important to avoid unnecessary usage of system memory. System overhead refers to the processing time required by system software, which includes the operating system and any utility that supports application programs. Some applications make the device non-responsive, so the total overhead to the system must be reduced.

4.1.2 Redundancy

Redundancy is important as it provides fault tolerance, but it can consume a lot of memory as it requires duplication of data set or error correcting code that has to be stored in memory. Normally, class data is stored in individual files, causing

too much memory usage and may result in lots of redundancy, example, in storing strings.

4.1.3 Parsing Data

Parsing checks for correct syntax and builds a data structures, mostly a parse tree or other hierarchical structure. On application launch, the classes are loaded in system memory and parsing is done that adds extra burden to the system performance.

4.1.4 Verification

Byte Code verification is important in order to verify whether the integrity of a code fragment is correct. As Java run-time system is unaware whether the code is trustworthy or not. Byte-code verification, if done at launch of an application, will cause the system to slow down.

4.1.5 Optimization

To save battery life, increase performance, and make system response faster, Dalvik perform byte-code optimization. Optimization is important in case of cell phones, as they have limited battery and resources for performing heavy computations (comparative to personal computers). The optimization is discussed in the next section in detail.

4.2 Dalvik optimizes and overcomes the issues stated of a typical VM's

To overcome memory limitations, system overhead, redundancy, and similar issues Dalvik performs optimization of byte code that involves following steps, which is incontrast a representation of capabilities of typical VM.

4.2.1 System Memory

To minimize the system memory usage, dex files are mapped read-only (for security purpose, that is discussed later in this report) and also sharing is allowed between processes. This avoids unnecessary repetition of data, and reduces memory usage.

4.2.2 Redundancy and Space

Dalvik overcomes the issue of memory space (consumed by redundancy, and separate files for each class) by aggregating the multiple classes into single dex file. This saves a lot of memory for the system.

4.2.3 System Overhead

System Overhead is caused due to "Just-in-Time" loading of classes in VM. JIT compilers have memory overhead because they need a code cache and supporting data structures. This could even make system non-responsive in some cases. For reducing overhead to the system the byte code is optimized by ordering the byte-code and word alignment adjustment, before launching an application. Dalvik is therefore optimized for running many concurrent instances even in the limited memory of a mobile phone [21].

4.2.4 Verification

As discussed above the byte-code verification is slow process, so we make processing fast by performing "pre-verification" of this byte code. This also reduces the system overhead that will be caused just on launching an application.

4.2.5 Parsing

As already discussed that parsing adds extra burden to the system, so the byte-code is re-written ahead of the time that is required by optimization method. It's like NO-JIT [10], which makes execution faster.

4.3 Overview of the operations performed by Dalvik

Application code is in the form of `.jar` or `.apk` files containing meta-data files. The classes.dex are extracted from archive before they can be used The dx tool converts Java byte-code to Dalvik byte-code. Other operations such as realignment, optimization and verification are performed on Dalvik byte-code.

4.3.1 Details

The first step is to create a `dex` file. The method is stated for DEX preparation followed by optimization [25].

1. Create a Dalvik-cache file in the Dalvik cache directory that is `$ANDROID_DATA/data/dalvik-cache`. Note: To work, create, delete and/or modify files in cache, appropriate privileges are required.
2. The `classes.dex` entry is extracted from the zip achieve.
3. For fast and easy access with the current system, memory mapping is performed, that includes byte-swapping and structure realigning. Moreover checks are made to ensure that data indices and file offsets fall within valid range. Note: These procedure make no logical changes to the DEX files.
4. In obtained ODEX (Optimized DEX file), pre-computed data is added to the header (at beginning of the file), this ensures space for the header which will be added later to it.
5. Now we come back to to verification and optimization process. But before that we must know about a program called *dexopt*. It is a program that creates an odex file but removes the `classes.dex` file in the apk. It performs VM initialization, loads DEX files from class path and then sets about verification and optimization. Its use will be clear after we discuss the verification and optimization process.
6. All of the classes are verified and optimized in DEX file by loading all of the classes in the VM and running through them. During the verification process certain things may fail to be verified or optimized which can cause a locking of resources. So this process is done in a separate VM, rather than on the VM on which the application is running. This whole procedure is done automatically by the dexopt described earleir.

7. Identification of illegal instructions before run time is required which involves verification, by scanning all the instructions in every method defined by every class in a DEX file.

8. Next comes optimization. A VM interpreter typically performs optimization the first time a piece of code is used, and achieved by replacing constant pool references with pointers to internal data structures. Operations that always work are replaced by its simpler forms. We discuss the Dalvik optimizer more in detail:

9. Functionality of Optimizer:
   - Replace method index with a vtable index (for method calls)
   - Replace field index with a a byte offset. (for instance field)
   - Replace handful of high-volume calls with "inline" replacements. (for string.length())
   - Cut any empty methods. Only should be called when objects are allocated to it.
   - Perform all pre-computation of data, will save heap space and computation time every time DEX file is loaded.

10. Dependencies of ODEX:

    The optimized DEX file have dependencies on:
    - All other DEX files in the bootstrap class path.
    - CRC-32
    - modification date from the originating classes.zip file entry.

    Dependencies list includes :
    - full path to dalvik-cache file
    - file's SHA-1 signature.
    - VM version number.

11. Generated DEX and future release:

    With some frameworks the heavy dexopt verification and optimization doesn't work well, so they have to rely on the ability to generate byte code and execute it. The support for this will be given in the future release [25].

4.4 Comparison of Dalvik VM with Java Virtual Machine

Performance of VM's vary with underlying operating system and hardware specifications and architecture. Here we will discuss Dalvik that is the chosen VM for Android systems and will make a brief comparison with JVM.

4.4.1 Memory Usage Comparison

JVM spends most of its resources on garbage collection and can not free more of its memory when required i.e., when an "Out of Memory" exception is thrown. Heap size (the memory space where the program lives) is calculated according to physical memory. JVM uses a large portion of the memory for runtime libraries created

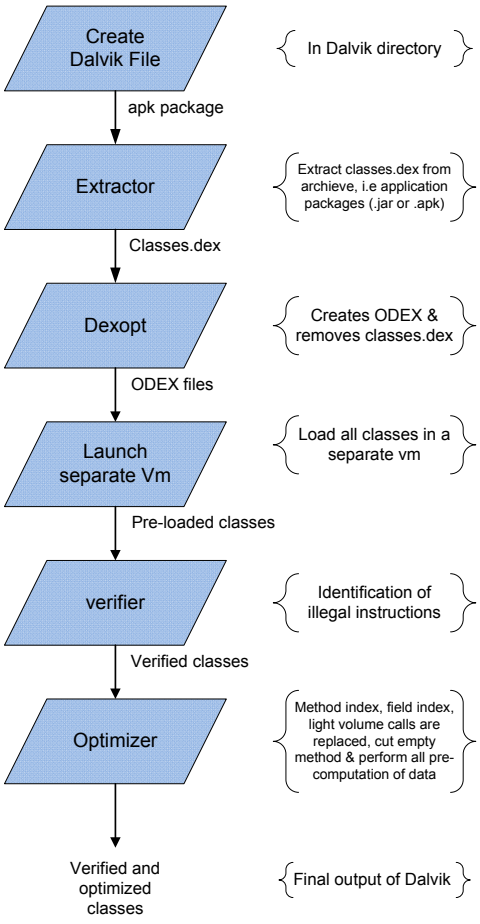


Figure 4.1: Representation of Dlavik Flow.

in shared memory. Compared to Dalvik, it operates on its own byte code that is optimized for low memory requirements. Also Dalvik file format allows direct execution Dalvik has an additional layer for security, including process separation and mapping file permissions to the underlying Linux platform. Dalvik is secured VM because it provides sharing of code between processes without giving it permissions to edit the shared code.apping file into memory, without the need to create a truckload of data structures just to be able to load the file.

4.4.2 Architecture Comparison

JVM is stack based, and operates on Java byte code. Stack based architectures are said to be slower as stack is stored in memory. Unlike other VM's , that are stack based, the Dalvik architecture is register based. Dalvik could be a very interesting VM even for use on the desktop, not just on the smaller devices. Register-based stack machines can be faster than stack-based ones. Fewer memory accesses also increase the speed of a register machine, and provide more opportunities for parallel execution during super scalar execution.

4.4.3 Multiple instance and JIT Comparison

JVM runs by default on JIT. This makes the loading of application too much slower for devices with limited resources such as cell phones. Dalvik is optimized to allow multiple instances of the VM to run simultaneously even in little memory. The main reason for that is ahead-of-time optimization, as we know that JIT is expensive and unpredictable where efficient execution is needed on limited resources.

4.4.4 Reliability Comparison

In current standard Java runtime systems, the failure of a single component can have significant impacts on other components. In the worst case, a malicious or erroneous component may crash the whole system. On the other hand Dalvik, runs every instance of VM in its own separate process. Separate process prevent all applications from crashing in case if the VM crashes [17].

4.4.5 Supported Libraries Comparison

The following table describes different libraries supported by Dalvik VM and Java VM.

Libraries	**Dalvik**	**Standard Java**
java.io	Y	Y
java.net	Y	Y
android.*	Y	N
com.google.*	Y	N
javax.swing.*	N	Y
...	...	...

Table 4.1: Supported Libraries Comparison [13]

4.4.6 Comparison Concluded

The following table depicts different aspects of both the VMs.

Criteria	**Dalvik**	**JVM**
Architecture	Register-based	Stack-based
OS Support	Android	All
Reverse Engineering-tools	a few(dexdump,ddx)	many (jad,bcel,findbugs,...)
Executables	DEX	JAR
Constant-Pool	per application	per class

Table 4.2: Different Aspects of both the VM's [13]

Chapter 5

Conclusion

Open source platforms have gained the attention of end users and businesses alike due to their free license and modifiable source. It became a challenge for the open source platforms to compete with the other platforms that had already gained popularity in market and had won the trust of manufacturers, developers and end users. The advancements to the Open Source platforms, encouraged Open Handset Alliance to launch Android mobile operating system to the market. Android is the first free, open source, and fully customizable mobile platform. It offers a full stack including an operating system, middle ware, and key mobile applications. It also contains a rich set of APIs that allows third-party developers to develop useful applications.

In this report we presented basic theme behind Dalvik. Dalvik is the Virtual Machine that is designed to work on Android operating system. When we talk about Open Source platform the first thing that comes in mind is Linux, as its free source, supporting multi-tasking, process threading, secure and ability to run on a system with limited resources. These factors became the motivation for Android to be build on Linux Kernel (version 2.6).

We included some facts and figures that shows how the Android market is gaining the attention of mobile market. Our report included some basics about virtual machines and then Dalvik VM in detail. Discussing the byte codes , conversion from Java byte code to Dalvik byte code, working of Dalvik, and many other aspects of Dalvik VM that could be of importance for a reader.

Comparison of Dalvik with Java virtual machine was discussed so that a clear picture could be presented to understand that in what ways the Dalvik is better than JVM. By studying the report it will easy for the reader to make a conclusion about the

VM that could be light weighted, performance efficient and more stable.

Our report provides a deep picture of the byte code that Dalvik uses for its operation. The whole scenario is presented regarding Dalvik's operation on its byte code. The report provides thorough understanding of Dalvik, which can generate ideas for the researcher to make Dalvik more efficient and faster. And to generate the Dalvik code in such a way that it loads more quickly then ever, hence making the execution of application even faster.

The comparison of the `dex` with Java compressed and uncompress file, we can observe how much less memory space it consumes. Execution of a light weight byte-code will make it faster than before. A research work to make the code more arranged and easy for VM to execute upon can bring a lot of performance up-gradation.

References

[1] Homepage: Android. Available at: http://www.android.com/.

[2] Homepage: Dalvik Virtual Machine. Available at: http://www.dalvikvm.com/.

[3] AdMob. Mobile advertising network. Available at: http://www.admob.com/.

[4] Admob Team. Admob mobile metrics: July 2009, report. Available at: http://metrics.admob.com/2009/08/july-2009-metrics-report/.

[5] Apache License. Apache license, version 2.0. Available at: http://www.apache.org/licenses/LICENSE-2.0.

[6] Dan Bornstein. Dalvik virtual machine: Internals. Available at: http://sites.google.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf.

[7] Daniel Bradby. Android market monthly revenue revealed. Available at: http://www.jtribe.blogspot.com/search/label/Android.

[8] Daniel Bradby. Top 10 grossing android apps. Available at: http://jtribe.blogspot.com/2009/10/top-10-grossing-android-apps.html.

[9] IBM Corporation. Ibm systems virtualization, version 2 release 1. Available at: http://publib.boulder.ibm.com/infocenter/eserver/v1r2/topic/eicay/eicay.pdf.

[10] Jeff Wilcox. Android performance 2: Loop speed and the dalvik vm. Available at: http://occipital.com/blog/tag/dalvik/.

[11] Justin Shapcott. Flurry stats show rush of android development. Available at: http://www.mobilemarketingwatch.com/flurry-stats-show-rush-of-android-development-4340/.

[12] Koji Hisano. A clean room implementation of android's dalvik virtual machine on java. Available at: http://code.google.com/p/android-dalvik-vm-on-java/.

[13] Marc Schonefeld. Reconstructing dalvik applications. Available at: http://cansecwest.com/csw09/csw09-schoenefeld.pdf.

[14] Motorolla. Droid motorolla: An android phone. Available at: http://www.motorola.com/Consumers/US-EN/Consumer-Product-and-Services/Mobile-Phones/Motorola-DROID-US-EN.

[15] Multiple Facets Inc. Hello im!: An android aol instant messenger. Available at: http://www.multiplefacets.com/application_aim.html.

[16] Nancy Gohring. Google, sun may clash over android's use of java. Available at: http://www.pcworld.com/article/139760/google_sun_may_clash_over_androids_use_of_java.html.

[17] Neil Bartlett. Google android, now 100% java-free. Available at: http://neilbartlett.name/blog/2007/11/13/google-android-now-100-java-free/.

[18] Network Dictionary. Computer programming software terms, glossary and dictionary. Available at: http://www.networkdictionary.com/software/b.php.

[19] OHA. Open handset alliance. Available at: http://www.openhandsetalliance.com/.

[20] Peter Haggar. Understanding java bytecode. Available at: http://www.ibm.com/developerworks/ibm/library/it-haggar_bytecode/.

[21] Ryan Slobojan. Dalvik, android's virtual machine, generates significant debate. Available at: http://www.infoq.com/news/2007/11/dalvik.

[22] Y. Shi, K. Casey, M.A. Ertl, and D. Gregg. Virtual machine showdown: stack versus registers. *ACM Transactions on Architecture and Code Optimization (TACO)*, 4(4):2, 2008.

[23] J.E. Smith and R. Nair. *Virtual machines: versatile platforms for systems and processes.* Morgan Kaufmann Pub, 2005.

[24] T. Suganuma, T. Ogasawara, M. Takeuchi, T. Yasue, M. Kawahito, K. Ishizaki, H. Komatsu, and T. Nakatani. Overview of the IBM Java just-in-time compiler. *IBM Systems Journal*, 39(1):175–193, 2000.

[25] The Android Open Source Project. Dalvik optimization and verification with dexopt. Available at: `http://www.netmite.com/android/mydroid/dalvik/docs/dexopt.html`.

[26] VMWarez. Vm warez: Where virtualization is reality. Available at: `http://www.vmwarez.com/2006/05/super-fast-server-reboots-another.html`.