

A Study on Android development kits and the ‘Phone Gap’ Framework

Sabyasachi Patra

Mukesh Patel school of Technology and Management, Computer science, Mumbai, India

Abstract: - The Android platform is a new generation of a smart mobile platform launched by Google. It is open-sourced and is based on Linux so that it developers can use it freely for development. It provides an easy-to-use development kit (the SDK) developed on the Android OS. There are different ways to go about Android programming right from hardware-oriented programming techniques to platform-independent ones. The paper is about various programming environments within the Android paradigm. It deals with their characteristics, advantages and differences between them.

Keywords: - *Android, Phone Gap, Web Application, SDK, NDK*

I. INTRODUCTION

Android software development is the process by which new applications are created for the Android operating system. Applications are usually developed in the Java programming language using the Android Software Development Kit. There are many environments to program an Android app and every single environment has its advantages and disadvantages.

Android Development Kit:

- Software Development Kit
- Native Development Kit

II. ANDROID SDK

The most common and obvious approach of programming Android-Apps is the Android SDK (Software Development Kit) by Google. It provides all the functionality to develop any kind of Android Application. We use the Eclipse IDE with the ADT (Android Development Kit) for programming in the SDK. The Android SDK is an approach that is suited for Java programmers, who want to develop applications exclusively for Android devices, and for every programmer with knowledge of object oriented programming or the time and motivation to learn this paradigm and the Java language. The SDK is essentially freeware, easy to install and allows developers to code using various API's. It provides you the libraries and developer tools necessary to build, test, and debug apps for Android. [1][2]

III. NDK

The Android NDK (Native Development Kit) is a companion tool or a toolset to the Android SDK that lets you build performance-critical portions of your apps in native code. It provides headers and libraries that allow you to build activities, handle user input, use hardware sensors, access application resources, and more, when programming in C or C++. [4][6] If we use the NDK and implement parts of the application in native code, the applications are still packaged into .apk file and they still run inside of a virtual machine on the device so that the fundamental Android application model does not change. It can also be defined as a bundle of tools, libraries and documentation that can be used to link in native C/C++ libraries into your Android Java projects. It provides you with everything you need to link your own C/C++ libraries to your apps, as well as commonly-used system libraries. [5]

Basically, this is the means by which parts of the Android application can be developed using code other than Java. This is possible through the Android NDK, which is available for three platforms: **Windows, Mac OS X and Linux 32/64-bit (x86)**. [7]

We need to balance the NDK's benefits against its drawbacks. Typical good candidates for the NDK are self-contained, CPU-intensive operations that don't allocate much memory, such as signal processing, physics simulation, and so on. Another area to warrant such tweaks is support for gaming and physics-based applications. When examining whether or not to develop in native code, we need to think about the requirements of the particular application and see if the Android framework APIs provide the functionality that is needed.

Required development tools for NDK:

- For all development platforms, GNU Make 3.81 or later is required. Earlier versions of GNU Make might work but have not been tested.
- A recent version of 'awk' (either 'GNU Awk') is also required.
- For Windows, Cygwin 1.7 or higher is required. The NDK will not work with Cygwin 1.5 installations.

The NDK includes a set of 'cross-toolchains' (compilers, linkers, etc.) that can generate native ARM binaries on Linux, OS X, and Windows (with Cygwin) platforms.

It provides a set of system headers for stable native APIs that are guaranteed to be supported in all later releases of the platform:

- libc (C library) headers
- libm (math library) headers
- JNI interface headers
- libz (Zlib compression) headers
- liblog (Android logging) header
- OpenGL ES 1.1 and OpenGL ES 2.0 (3D graphics libraries) headers
- libjnigraphics (Pixel buffer access) header (for Android 2.2 and above).
- A Minimal set of headers for C++ support
- OpenSL ES native audio libraries
- Android native application APIS

Advantages and uses of NDK:

- The use of native code with the help of android NDK is usually implemented for improving speed of performance.
- It is also intended to support the reuse of existing code and is a way to embed native libraries into an application package file (.apk) that can be deployed on Android devices.
- Apps can be written using the Android framework and then the JNI (Java Native Interface) can be used to access the APIs provided by the Android NDK. The JNI enables code running in a Java Virtual Machine to call libraries written in other languages, such as C. This technique allows us to take advantage of the convenience of the Android framework, but still allows us to write native code when necessary.
- Within an application, activity lifecycle call-backs such as onCreate(), onPause() and onResume() should be handled in native code. Applications that use such native activities must be run on Android 2.3 (API Level 9) or later as certain features of the Android framework will not be available in previous versions of android.

There are two main reasons to consider using the Android NDK

- **Performance**
- **Porting/Cross-platform.**

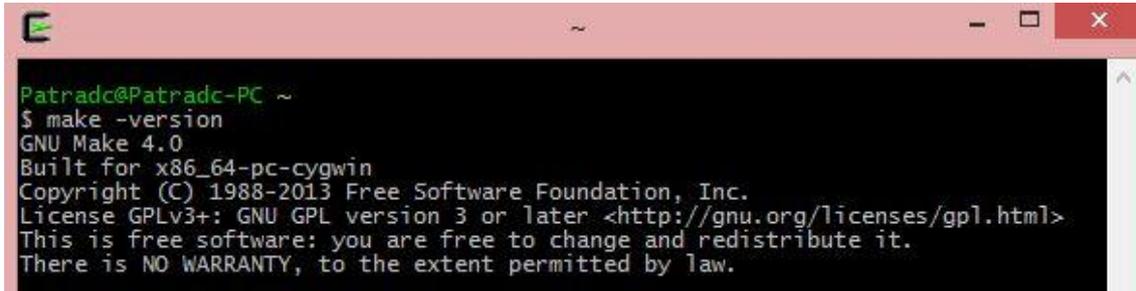
For performance, the NDK can provide high-performance operations for your application if it's truly processor bound. That is, the application algorithms are tapping into all of the device's processing power within the Dalvik VM so that the code may see performance benefits from running natively.

- Thus the native code can be used to optimize the application for speed especially when bit/byte operations are used in the code, like when we need to do compression/decompression of image files etc. Hence most games use the native C code libraries for 2D / 3d graphics, input, sound etc. [7]
- Running native code is complicated by the fact that Android uses a non-standard C library (libc, known as Bionic). The graphics library that Android uses to arbitrate and control access to the device is called the Skia Graphics Library (SGL), and it has been released under an open source license. Skia has back -ends for both win32 and 'Unix', thus allowing the development of cross-platform applications. [8]
- NDK can be used to leverage existing code that is already in C or C++. Here, using the NDK could simplify the application development, at the same time allowing for shared code with other platforms, as well as generally helping avoid maintaining parallel codebases in multiple programming languages. This perk is often leveraged with applications that target multiple operating systems using OpenGL ES.

Compiling and deploying NDK sample applications (The first step to NDK Development):

1. Open a command-line prompt (or Cygwin prompt on Windows)
2. Go to hello-jni sample directory inside the Android NDK. All the following steps have to performed from this directory: **\$ cd \$ANDROID_NDK/samples/hello-jni**

3. Create Ant build file and all related configuration files automatically using android command (android.bat on Windows). These files describe how to compile and package an Android application: **android update project -p** .
4. Compile, package, and install the final 'HelloJni' APK (an Android application package). All these steps can be performed in one command, thanks to Ant build automation tool. Among other things, Ant runs 'javac' to compile Java code, AAPT to package the application with its resources, and finally ADB to deploy it on the development device. Following is only a partial extract of the output: **\$ ant install**



```
Patradc@Patradc-PC ~
$ make -version
GNU Make 4.0
Built for x86_64-pc-cygwin
Copyright (C) 1988-2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

Fig.1: Cygwin Prompt on Windows

IV. PHONEGAP

To develop an application with Phone Gap there has to be a Software Development Kit (SDK) that fits the end device because Phone Gap can't run stand-alone. It grants access to native device APIs. It is a good choice for Web Application developers who want independence from a web server and/or want to distribute their application. 'PhoneGap' currently supports development for the operating systems Apple iOS, Google Android, Microsoft Windows Phone (7 and 8) and Nokia Symbian OS. It uses HTML5 and CSS3 for visual rendering and JavaScript for the logic. [3]

Developing apps for multiple mobile platform needs a resource too many and a developer needs a firsthand knowledge of multiple IDEs and languages such as C++, Java, Objective C, etc. Cross-platform mobile application development is a fast and efficient way to make available an application on all mobile technologies with limited added cost and software resources. The main challenge in cross-platform mobile application development is the OS fragmentation. The trend of increased fragmentation coincides with the growing number of mobile platforms. First, there were BlackBerry, Symbian and Bada smartphones, then came the powerful iPhone and Android platforms, HP came with WebOS and then Microsoft introduced the Windows Phone. Now Intel, Samsung and various other companies under the open source alliance have forged a new mobile OS called Tizen which comes out in 2014 on selected devices. This means that companies have to keep launching new products to make their presence felt on all mobile platforms.

Different programming languages are required for different mobile platforms, so Mobile OS fragmentation is the worst and most diverse fragmentation of all. This adds a sizable technical challenge to launching mobile applications on all mobile platforms. There are also a few other challenges for the developer like consistent user experience across all platforms. [3][4]

- Phone Gap is a framework that uses web technology to realize native mobile application, such as iPhone or Android
- It uses HTML, CSS, JavaScript technology with native characters like Camera, Media.
- It is the only open source frame to support seven platforms, which provides many interfaces to support native API through the calling of JavaScript.
- The user interface of the applications are made using HTML and CSS while the backend coding or the calculations can be called using JavaScript for interactivity, dynamism, and interaction with native-OS functionality.
- The final output is one that has features of both a web application and a native application. Certain features, such as Image Capture, NFC or Android Open-Accessory, may be implemented natively since there is currently no way to do this in JavaScript.
- The application logic and the UI are implemented using web technologies to allow for a consistent and unique user experience across devices.

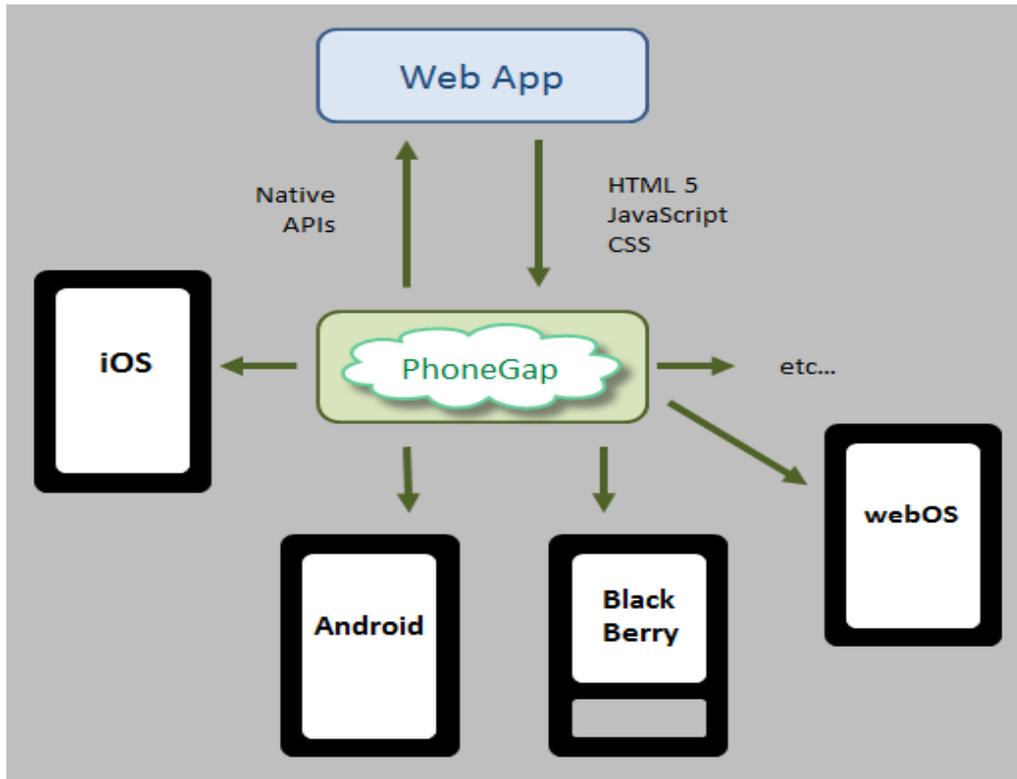


Fig.2: The role of PhoneGap

V. WEB APPLICATIONS (HYBRID)

These are applications that run on browsers. The application itself and its data are usually stored and mainly executed on a webserver. The user starts the 'WebApp' by sending a HTTP-request via a browser to the webserver where the application is stored. The webserver then generates or subsequently loads the HTML source code of the website and sends it back to the browser. Then the user is able to interact with the graphical user interface (GUI) of the 'WebApp'. It is represented by a website which runs in a browser. It is platform independent and written in web-based languages. It runs on mobile devices and also on every other device which have access to the internet (if the app is stored on a server) and a browser. These are also known as universal Mobile Applications and are of a great use to 'not- so – tech – savy' normal cell phone users. It takes **user – friendliness to an all new level.**

VI. A COMPARATIVE STUDY

Table I: Comparison

OPTION	SDK	NDK	Phone Gap	Web App
Installation	+	--	0	0
Support	++	+	0	+
Learn ability	-	--	0	0
Platform Independence	-	-	++	++
GUI creation	+	-	+	+
Potency	++	++	0	--
Performance	+	++	0	-
Debugging	++	+	+	--

V. CONCLUSION

Normal Android application development in both the industry as well as for personal purposes is done on the Android SDK using the various API's provided by Google and Android. The NDK is used for special purposes whenever required, as in for Physics simulations or certain kind of games. The depth of research carried out in the 'Phone Gap' and SDK programming domains are far wider as compared to the NDK applications and usage. Therefore this paper is a start towards our study and research on the NDK model of Android.

The basic purpose of our paper was to carry out a comparative study and review the existing SDK programming paradigms along with the 'Phone Gap' Hybrid application framework. We have also given an overview of the NDK, its importance, its usage and an overall picture of what it takes to install and use it.

ACKNOWLEDGMENT

This research paper is made possible through the help and support of many people both inside and outside the domain of Engineering and Science.

Especially, please allow me to dedicate my acknowledgment of gratitude towards our college Librarian Mr. Pradip Das and his team. I would also like to thank Mr. Anand Gawadekar of the NMIMS IEEE Committee due to which I could get all requested references seamlessly without any trouble and on time.

A sincere thanks to our college and Computer Science department H.O.D. Professor Dharendra Mishra for allowing us to enter the invaluable field of research in our so important final year of B.Tech. Finally, I would like to thank my parents who always encouraged me to do as much research I could do in my capacity in the final year and extend an outside support whenever and wherever required.

REFERENCES

- [1]. M. H. Amerkashi. Apptomarket. <http://code.google.com/p/apptomarket/>, Apr. 2013.
- [2]. S. Baltes. Run anywhere. web & mobile DEVELOPER, 06:94–100, May 2013
- [3]. PhoneGap. Phonegap api documentation. <http://phonegap.com>, Mar. 2013.
- [4]. S. Ratabouil. Android NDK Beginner's Guide. Packt Publishing, 2012.
- [5]. A. Sarah, V. Graupera, and L. Lundrigan. Pro Smartphone Cross- Platform Development: iPhone, Blackberry, Windows Mobile and Android Development and Distribution. Apress, 2010.
- [6]. X. Shu, Z. Du, and R. Chen. Research on mobile location service design based on android. In 5th International Conference on Wireless Communications, Networking and Mobile Computing, 2009. WiCom'09, pages 1–4, 2009.
- [7]. K.-C. Son and J.-Y. Lee. The method of android application speed up by using NDK. In 2011 3rd International Conference on Awareness Science and Technology (iCAST), pages 382–385, 2011.
- [8]. J. Stark. Android Apps mit HTML, CSS und JavaScript. O'Reilly, 1 edition, Dec. 2010.
- [9]. C. Ullenboom. Java ist auch eine Insel - programmieren mit der Java Platform, Standard Edition 6. Galileo Press, Bonn, 7. aufl. edition, 2008