International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 4 Issue: 6 438 - 440

Smart mobile service that helps users to access the applications faster

Mr. Amit Rohra,
Post Graduate, MCA,
Vivekananda Education Society's
Institute of Technology,
Collector's Colony, Chembur.
Mumbai - 400 074
amit.rohra@ves.ac.in

Mr. Nikesh Posam
Post Graduate, MCA,
Vivekananda Education Society's
Institute of Technology,
Collector's Colony, Chembur.
Mumbai - 400 074
nikesh.posam@ves.ac.in

Prof. Nishi Tiku
Senior Lecturer, H.O.D,
Vivekananda Education Society's
Institute of Technology,
Collector's Colony, Chembur.
Mumbai - 400 074
nishi.tiku@ves.ac.in

Abstract:-Our target, today's generation, we are targeting the root of today's generation i.e. Mobile phones (Smart phone). Our intent is to engineer a service that will be running on operating system to access the application.

Keywords:-*App, Activity, Drawer, Layout, Coordinates, Service*

*****

1. Introduction

In today's generation smart phones have become one essential part of human being. We use the mobile phone throughout the day. There are numerous application software's have been developed for mobiles operating system to get most out of the mobile phone. Our concept is to provide the user a quicker access to the application installed on the system. Our main focus is on Android operating system as it is the only operating system that has larger number of users as compared to others but we can process the same service for other operating system like Windows, iPhone & Blackberry OS.

2. Android

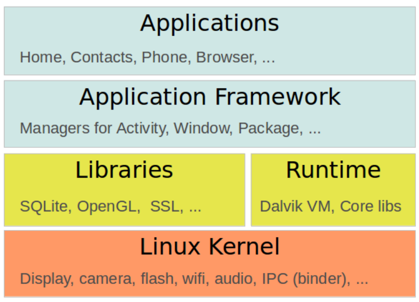
Android is a mobile operating system developed by Google. The first version was brought to market in 2008. Since then Android is experiencing a great increase in market share. It is largely open-source and is based on Linux 2.6. Application programming is done in Java but is being compiled to so called DEX bytecode and is being run by the Dalvik Virtual Machine (DVM). This section describes fundamentals of Android including system architecture and runtime environment.

System Architecture

Android uses a 4-layered system architecture depicted in figure. The layers are described in the following**.**

2.1.1 Linux Kernel

Provides low level operating system functionality such as memory management, security, process management etc. It also facilitates all the hardware drivers.


Applications
Home, Contacts, Phone, Browser, ...
Application Framework
Managers for Activity, Window, Package, ...
Libraries
SQLite, OpenGL, SSL, ...
Runtime
Dalvik VM, Core libs
Linux Kernel
Display, camera, flash, wifi, audio, IPC (binder), ...

2.1.2 Android Runtime

It includes many libraries mostly written in the Java programming language, which provide a lot of the functionality of the Java core libraries. The Android Runtime layer also offers the Dalvik Virtual Machine (DVM) which is being described in 2.2.

2.1.3 Libraries Provides

C/C++ libraries which are used by many other components within Android. Those libraries are e.g. the System C library, SQLite, 3D libraries and so on.

2.1.4 Application Framework

Android offers an extensive framework enabling application to easily communicate and share data with each other. Furthermore the framework simplifies accessing systems resources such as hardware, location data and background services.

2.1.5 Applications

On this layer Android applications accessible by the user are located. Android comes with some standard applications such as Contacts, Messaging etc.

Dalvik Virtual Machine

Android applications are developed using the Java programming language. Nevertheless you cannot make use of the full Java framework since Android ships with its own framework. Java language features such as Reflection are not supported by this framework due to performance issues. The Dalvik Virtual Machine (DVM) is a virtual machine which is optimized for (low-resource) mobile devices. Unlike the virtual processor model of the Java Virtual Machine (JVM) the DVM's processor model makes use of machine registers thus taking advatage of modern mobile CPU's. In addition bytecode for the DVM (called DEX bytecode1) is much smaller than Java bytecode and can be executed much faster. How a Java source file becomes DEX bytecode is described in the following and illustrated in figure 2. Given a Java source file (*.java) the Java compiler javac creates a Java bytecode file (*.class). Now the dx tool transforms it to a *.dex file which is capable of being executed by the DVM.

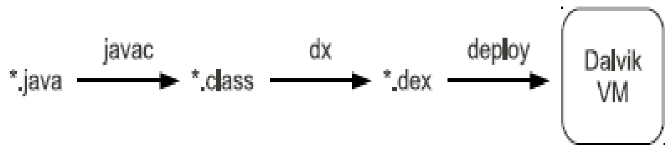


Fig. 2. From Java sourcecode to DEX bytecode

Android Applications

Android applications consist of DEX bytecode, resources and data and are deployed as *.apk files. They are subject to a lifecycle and are executed in a so called “Sandbox". That is every application runs in its own DVM in an independent Linux process. When an application is being installed to an Android device it is being assigned a new user and group ID. Concerning security there is no possibility to illegally access another process or its data since the user and group IDs are different.

3. Android's Component System

Before going into the details of Android's component system I first want to introduce what a software component really is. In 1996 the European Conference on Object-Oriented Programming (ECOOP) defined it as follows:

“A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."

In short the following defines a component:

1. contractually specified interface
2. explicit context dependencies
3. independent deployment

Components must conform to a component model which specifies form and properties of a component as well as how components interact with each other and how they can be combined.

Components in Android

Android allocates four different types of components:

Activities are suitable for interacting with the user and presenting the user interface.

Services can be used to run background tasks. They do not have a user interface.

Content Providers are suited for providing data to other applications. Thus it is possible to “break out" of the Sandbox and provide data through a defined interface.

Broadcast Receivers are used to receive broadcasts either from the Android system or from other applications.

Communication between Components

As a major aspect of a component model is the interaction between components this subsection is to clarify how it is implemented in Android. The communication between components in Android is done by using Intents and Intent Filters. The calling component uses Intent to announce the wish for communication with another component. The called component has an Intent Filter to receive the Intent whereat the Intent must match the conditions of the Intent Filter.

Intent Filters

Intent Filters are defined in a manifest _le called AndroidManifest.xml. An example is shown following.

```
<receiver android:name="org.example.MyReceiver">
      <intent - filter >
            <action android:name="org.example.TEST" />
     </ intent - filter >
 </ receiver >
```

The XML tag <receiver> stands for a Broadcast Receiver. This Broadcast Receiver is annotated with an Intent Filter whose action name is org.example.TEST meaning the Broadcast Receiver is able to process actions with that name. So we have an explicitly defined interface through which the component can be used. That applies to the first point of the definition of a component.

Intents

Intents Amongst other information Intents contain information about the desired target component. Intent can either be explicit or implicit. Explicit means that the target component is known by its Java name. Implicit means that the target component is not known but the desired action is. In case of an implicit Intent the Android system is responsible for finding a component suitable for executing the action. If more than one component is found the Android system asks the user interactively which one to choose. If none is found Android will display an error message.

```
/* explicit intent */
 Intent ei = new Intent ( org. example . MyReceiver . class );

 /* implicit intent */
 Intent ii = new Intent (" org. example . TEST ");
```

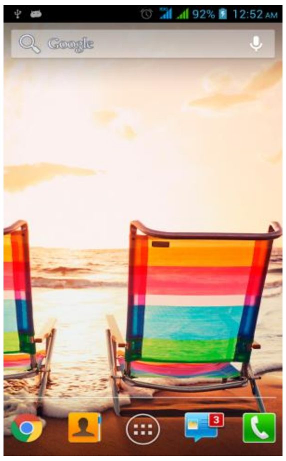
4. Concept

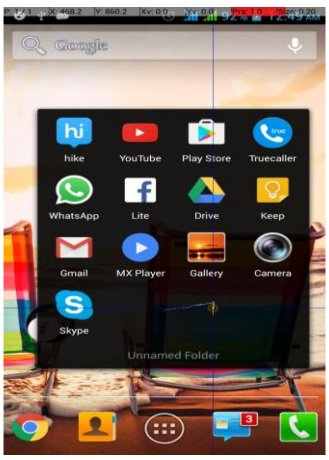
Traditionally user to open any application user needs to open the application drawer, then swipe to the screen where the application is present and then click on the application to launch it. These steps won’t trouble you if you have few screens in the drawer (i.e. less applications installed). But now a days single user have many number of applications that leads to many screens in application drawer, so to launch the application user has to swipe n screens to locate the application and it is also a brain teaser for users to remember the screen & location of each & every application.

While implementing the proposed daemon, user need not to open the application drawer & swipe the screens to locate the application, rather user just have to tap the location on the Home screen of mobile phone, now this tap location will query the application drawer & fetch links to the applications installed at tap location in every screen & pops up a window which will display the shortcuts to the applications present at the location.

5. Proposed UI

Following images shows the proposed UI for the mobile phones:





As shown in the picture we can get the location of the screen when w tap on the screen, subsequently we can pass this coordinates location to the application drawer and fetch the links to applications present at the supplied position from all the screens. These links will be used to create the invisible shortcut on the home screen. So when user clicks on the screen he gets the direct access to the application from home page without going and swiping the application drawer.

6. Conclusions

Opening the drawer, browsing and locating the application requires more user efforts and power consumption in traditional operating system. By implementing this service in the mobile operating system, user gets the faster access to the application and he don't need to remember the location of the each and every application. This service can be integrated with operating system services so it will not take more power in terms of battery consumption which will increase the battery performance to some extent.

7. Acknowledgment

This research was supported by Vivekananda Education Society's Institute of Technology. We are thankful to our colleagues who provided expertise that greatly assisted the research, although they may not agree with all of the interpretations provided in this paper. We are also grateful to Prof. Nishi Tiku for assisting with Mobile Application Development and moderated this paper. We have to express out appreciation for sharing their pearls of wisdom with us during the course of this research.

8. References

[1] Rick Boyer, Kyle Mew Android Application Development Cookbook .
[2] https://homepages.thm.de/~hg51/Veranstaltungen/MasterSeminar1011
[3] https://developer.android.com/reference/android/app/Service.html
[4] Jakob Iversen, Michael Eierman Learning Mobile App Development (A hands-on guide to develop i-os and android application).
[5] Michael Burton Android App Development for Dummies.
[6] Marko Gargenta & Masumi Nakamaru Learning Android.