ISSN NO. 2320-5407

Journal homepage: http://www.journalijar.com
Journal DOI: 10.21474/IJAR01

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH

RESEARCH ARTICLE

An Enhanced Systematic Approach for Changing Mobile Settings and Detecting Mobile Device.

***M.Asan Nainar[1] , G.Dharani Devi[2] and K.Ganesh[3].**

1. Assistant Professor, Dept. of Computer Applications,Valliammai Engineering College,Tamil Nadu, India.
2. Assistant Professor, Dept. of Computer Applications,Valliammai Engineering College,Tamil Nadu, India.
3. PG Student, Department of Computer Applications, Valliammai Engineering College, Tamil Nadu, India.

Manuscript Info

Manuscript History:

Received: 12 April 2016
Final Accepted: 19 May 2016
Published Online: June 2016

Key words:
Profile, Registration, Location and Mode

****Corresponding Author***

M. Asan Nainar.

Abstract

Android are the latest and a rapid growing technology available for all the users or customers in today market. New mobile devices, technologies, methods, applications are introduced every day. The main aim of this proposed technique is systematically change the mobile settings like Mobile Mode, Bluetooth, Wi-Fi, etc. The feature of the application is finding the missed mobile which is in silent mode. Create a profile and set the mode as Wi-Fi is on, Silent mode is on. While creating a new profile based on location, the device must be present in the particular location. Once the profile is created, it starts working without user interaction and physical presence. It can be achieved through sending SMS approach.

Introduction:-

Android platform is selected for development because it is currently one of the most widely adopted smartphone platforms and it is open for research. Applications running on Android are primarily written in Java programming language. An Android application is first compiled to Java virtual machine compatible .class files that contain Java byte code instructions. These .class files are then converted to Dalvik virtual machine executable .dex files that contain Dalvik byte code instructions. Finally, the .dex files are encapsulated into an Android application package file (i.e., an .apk file) for distribution and installation. For ease of presentation, we in the following may simply refer to "Android application" by "application" when there is no ambiguity. An Android application typically comprises four kinds of components as follows:

- Activity: Activities are the only type of components that contain graphical interfaces to display information and interact with users. An application may comprise multiple activities that work together to provide a cohesive user experience. An activity can be launched from other activity or service components.
- Service: Services are components that run in the background for conducting long-running tasks (e.g., sensor reading or data synchronization etc.). Other components, typically activities, can start and interact with services.
- Broadcast receiver: Broadcast receivers define how an application responds to system-wide broadcast messages sent from other components, applications or the Android system. It can be statically registered in an application's configuration file (e.g., AndroidManifest.xml), or dynamically registered in an activity or service component at runtime.
- Content provider. :Content providers manage shared application data persisted in file systems, databases or network locations. They provide an interface for other components or applications to query or modify these data.

Background Theory:-

Android Studio:-

Android Studio is an integrated development environment (IDE) for developing for the Android platform. It was announced on May 16, 2013 at the Google I/O conference by Google's Product Manager, Katherine Chou. Android Studio is freely available under the Apache License 2.0.

Android Studio was in early access preview stage starting from version 0.1 in May 2013, then entered beta stage starting from version 0.8 which was released in June 2014 The first stable build was released in December 2014, starting from version 1.0

Based on JetBrains' IntelliJ IDEA software, Android Studio is designed specifically for Android development it is available for download on Windows, Mac OS X and Linux, and replaced Eclipse Android Development Tools (ADT) as Google's primary IDE for native Android application development.

Features:-

- Flexible Gradle-based build system.
- Build variants and multiple apk file generation.
- Code templates to help you build common app features.
- Rich layout editor with support for drag and drop theme editing.
- The lint tools to catch performance, usability, version compatibility, and other problems proGuard and app-signing capabilities.
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine.

Debug And Performance:-

Android Virtual Device (Avd) Manager:-

AVD Manager has updated screens with links to help you select the most popular device configurations, screen sizes and resolutions for your app previews. Therefore The AVD Manager comes with emulators for Nexus 6 and Nexus 9 devices and also supports creating custom Android device skins based on specific emulator properties and assigning those skins to hardware profiles. Android Studio installs the Intel® x 86 Hardware Accelerated Execution Manager (HAXM) emulator accelerators and creates a default emulator for quick app prototyping.

Memory Monitor:-

Android Studio provides a memory monitor view so you can more easily monitor your app's memory usage to find deallocated objects, locate memory leaks and track the amount of memory the connected device is using. With your app running on a device or emulator, click the Memory Monitor tab in the lower right corner to launch the memory monitor.

Log Messages:-

When you build and run your app with Android Studio, you can view adb and device log messages (logcat) in the DDMS pane by clicking **Android** at the bottom of the window. If you want to debug your app with the Android Debug Monitor, you can launch it by clicking **Monitor** in the toolbar. The Debug Monitor is where you can find the complete set of DDMS tools for profiling your app, controlling device behaviors, and more. It also includes the Hierarchy Viewer tools to help optimize your layouts.

Android Development Tools:-

Android SDK:-

The Android Software Development Kit (Android SDK) contains the necessary tools to create, compile and package Android applications. Most of these tools are command line based. The primary way to develop Android applications is based on the Java programming language.

Android debug bridge (adb):-

The Android SDK contains the Android debug bridge (adb), which is a tool that allows you to connect to a virtual or real Android device, for the purpose of managing the device or debugging your application.

Android Developer Tools And Android Studio:-

Google provides two integrated development environments (IDEs) to develop new applications. The Android Developer Tools (ADT) are based on the Eclipse IDE. ADT is a set of components (plug- ins), which extend the Eclipse IDE with Android development capabilities. Google also supports an IDE called Android Studio for creating Android applications. This IDE is based on the IntelliJ IDE.

Both IDEs contain all required functionality to create, compile, debug and deploy Android applications. They also allow the developer to create and start virtual Android devices for testing. Both tools provide specialized editors for Android specific files. Most of Android's configuration files are based on XML. In this case these editors allow you to switch between the XML representation of the file and a structured user interface for entering the data. Dalvik Virtual Machine the Android system uses a special virtual machine, i.e., the Dalvik Virtual Machine (Dalvik) to run Java based applications. Dalvik uses a custom bytecode format which is different from Java bytecode. Therefore you cannot run Java class files on Android directly; they need to be converted into the Dalvik bytecode format.

Android Runtime (Art):-

Google introduced the Android RunTime (ART) as optional runtime for Android 4.4. It is expected that versions after 4.4 will use ART as default runtime. ART uses Ahead of Time compilation. During the deployment process of an application on an Android device, the application code is translated into machine code. This results in approx. 30% larger compile code, but allows faster execution from the beginning of the application. 4.1.3 Security and Permission Concept in Android.

Java:-

Java is a very popular programming language developed by Sun Microsystems (now owned by Oracle). Developed long after C and C++, Java incorporates many of the powerful features of those powerful languages while addressing some of their drawbacks. Still, programming languages are only as powerful as their libraries. These libraries exist to help developers build applications.

Some of the Javas important core features are:

- Its easy to learn and understand.
- Its designed to be platform-independent and secure, using virtual machines.
- Its object-oriented. Android relies heavily on these Java fundamentals.

The Android SDK includes many standard Java libraries (data structure libraries, math libraries, graphics libraries, networking libraries and everything else you could want) as well as special Android libraries that will help you develop awesome Android applications.

Platform Independence Importance:-

With many programming languages, you need to use a compiler to reduce your code down into machine language that the device can understand. While this is well and good, different devices use different ma- chine languages. This means that you might need to compile your applications for each different device or machine language in other words, your code is not very portable. This is not the case with Java. The Java compilers convert your code from human readable Java source files to something called bytecode in the Java world. These are interpreted by a Java Virtual Machine, which operates much like a physical CPU might operate on machine code, to actually execute the compiled code. Although it might seem like this is inefficient These efforts have paid off in that Java performance in generally second only to C/C++ in common language performance comparisons. Android applications run in a special virtual machine called the Dalvik VM. While the details of this VM are unimportant to the average developer, it can be helpful to think of the Dalvik VM as a bubble in which your Android application runs, allowing you to not have to worry about whether the device is a Motorola Droid, an HTC Evo, or the latest toaster running Android. You dont care so long as the device is Dalvik VM friendly and that's the device manufacturers job to implement.

Let's take this bubble idea a bit further. Because Java applications run within the bubble that is a virtual machine, they are isolated from the underlying device hardware. Therefore, a virtual machine can encapsulate, contain, and manage code execution in a safe manner compared to languages that operate in machine code directly. Each Android application runs on the (Linux- based) operating system using a different user account and in its own instance of the Dalvik VM. Android applications are closely monitored by the operating system and shut down if they don't play nice Therefore, it's important to develop applications that are stable and responsive.

Java Pathfinder:-

Java PathFinder (or JPF for short) is a highly customizable execution environment designed for verifying Java byte-code programs. As shown in Figure 1, JPF takes as input the byte codes of the system under analysis, and a set of configurations specifying the properties to check. JPF systematically analyzes the target system starting from an entry point and checks for violations of the specified properties.

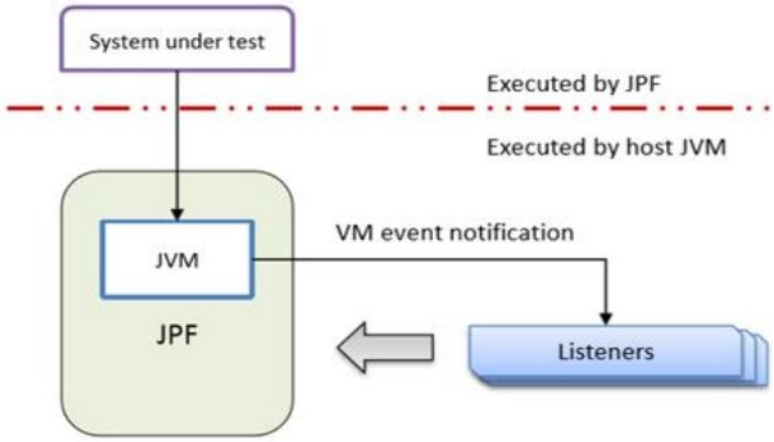

Fig 1: JPF listener mechanism

Proposed work:-

In this proposed system for analyzing systematically change the mobile settings like (Mode, Bluetooth, Wi-Fi, etc.). The feature of the application is finding the missed mobile which is in silent mode. Create a profile and set the mode as (mobile data is off, Silent mode is off etc.). It could create multiple profiles and change the airplane mode through message .

Proposed System Architecture and Methodology:-

System architecture:-

While creating a new profile based on location, the device must be present in the particular location. For Ex: If user wants to create a college profile, the device must be in college campus. Once the profile is created, it starts working without user interaction and physical presence. This works using service which is a component that runs in the background. This application needs an one time registration like user name, password and phone number. After the registration we could create different profiles and change profile settings from any mobile number using message command with password. In case of having mobile missed in silent mode, we will not be able to find without ring tone sound. In this case we can change the mode to general or some other sound profile from another mobile and find our mobile easily.

The following figure 2 depicts the system architecture. This application uses function oriented design approach. Every module and sub modules are made based on their functionality. These modules are designed and implemented separately and then they are integrated together to form the desired application.

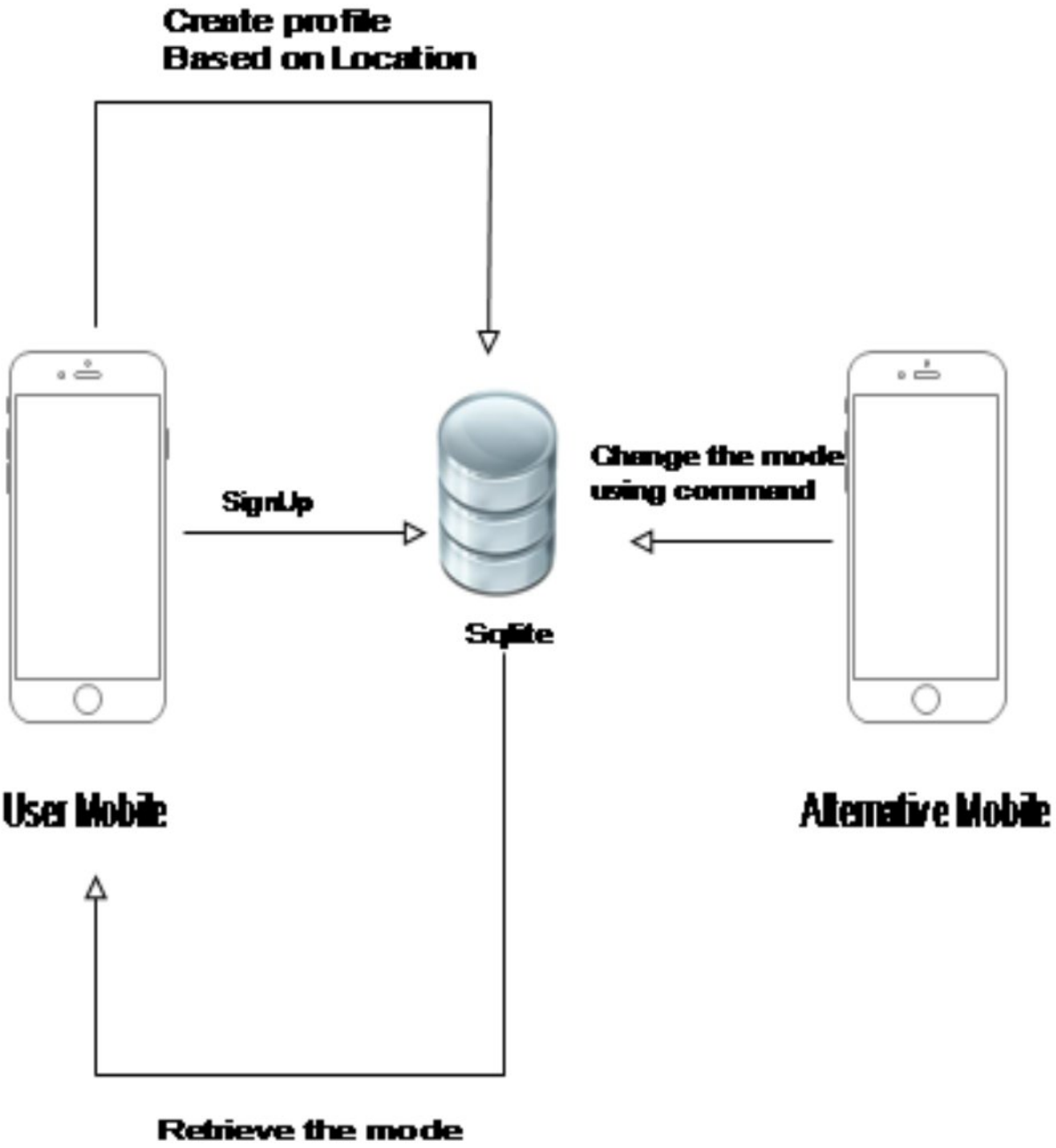

Fig 2 System architecture

Methodology Description:-

One time registration:-

In the first step, the user sign up into the location where the sign up process is based on one time registration. The user gives the user name , password, mobile number and the alternative mobile number.

Create the profile:-

In the second step, after signing up into the application the user profile is created and it is stored in the local database. The service runs in the background after completion of user sign up. user can change the profile mode based on the location.

Get Location:-

The profile mode to that particular location is picked from the database. These details are previously stored by the user. If the user enter into the new location, application get that location and our database checks the value whether

the user have set the mobile's profile mode to that location. If yes, it get the profile mode.

Automatic profile Change:-
After getting that particular user's current location and the profile mode to that location it automatically updates the profile mode of the mobile. This profile change is based on location.

Update modification:
In this module, the user can update and modify the profile change to which the user already have set the profile mode. So, it is possible to the user to update the profiles also.

Change the mode using message:-
In the final step, the user can change the profile mode. If the mobile is missed in the silent mode, user can change the silent mode to general mode by using the alternative mobile number.

Results and Discussion:-
The testing performance of the proposed work depicted in the following figure 3 and figure 4a, 4b and 4c summarizes the testing framework:

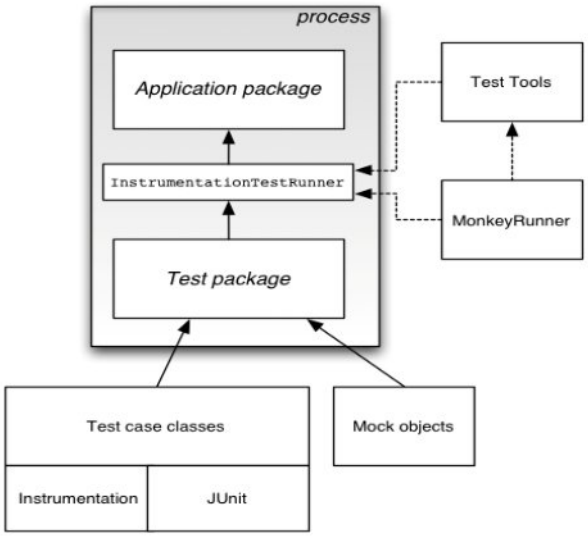

Fig 3 Testing Framework

To determine whether or not the application has met the requirement specifications. If profile is matched based on location then mobile mode will change automatically. GET LOCATION and AUTOMATICALLY PROFIE CHANGE performance verified separately. Android provides two Context classes that are useful for testing: Isolated Context provides an isolated Context, File, directory, and database operations that use this Context take place in a test area. Though its functionality is limited, this Context has enough stub code to respond to system calls. This class allows you to test an application's data operations without affecting real data that may be present on the device.

The Android testing framework returns test results back to the tool that started the test. If you run a test in Eclipse with ADT, the results are displayed in a new JUnit view pane. If you run a test from the command line, the results are displayed in STDOUT. In both cases, you see a test summary that displays the name of each test case and method that was run. You also see all the assertion failures that occurred. These include pointers to the line in the test code where the failure occurred. Assertion failures also list the expected value and actual value.

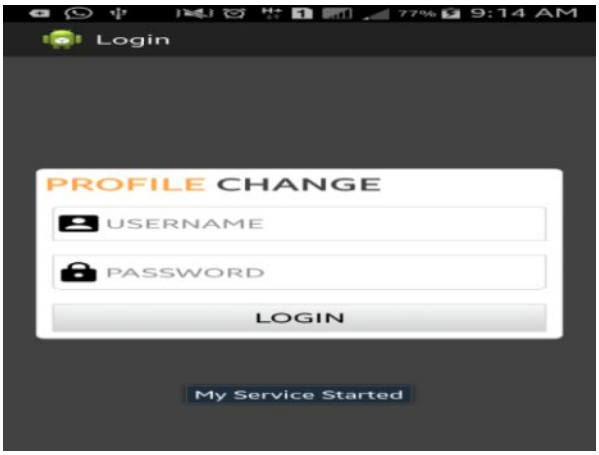

Fig 4a Login

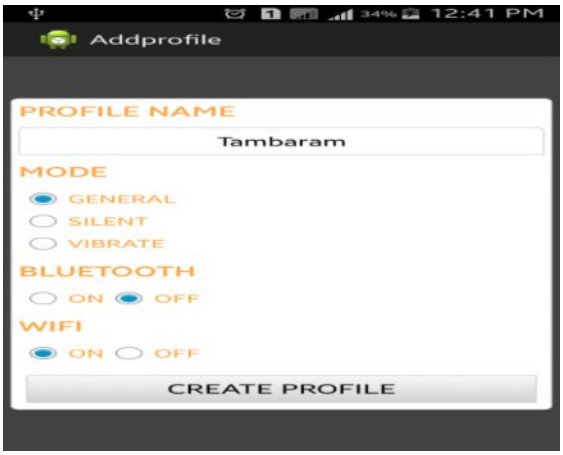

Fig 4b Profile

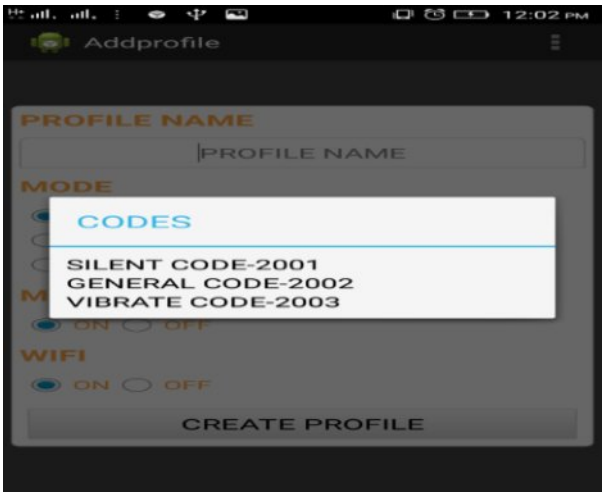

Fig 4c Profile Codes

Conclusion:-

In this paper, define the various multiple profiles. While creating a new profile based on location, the device must be present in the particular location to detect mobile mode change and missed mobile. In this application user can create multiple profiles but user cannot delete the profiles. In future introduce to add or remove options for deleting profiles. User can able to change mobile data also.

References:-

1. **Bharti Ahuja, Mayuri Deshmukh, Ruchika Borhade, Pooja Nikam** " Location Based Automatic Profile Changer and Mobiminder" , International Journal of Emerging Technology and Advanced Engineering, Volume 3,Issue 4,April 2013, pp.421-425.
2. **Bill Phillips** "Android Programming" The Big Nerd Ranch Guide,2nd Edition, 2016.
3. **Joel Murach's** "Android Programming", 2nd Edition, 2015.
4. "AndroidActivityLifecycles."URL:http://developer.android.com/ guide/components/activities.html.
5. "Android Developers Website." URL: http://developer.android.com/ index.html.
6. "Android Emulator."URL:http://developer.android.com/tools/help/ emulator.html.
7. "AndroidPlatformVersions."URL:http://developer.android.com/ about/dashboards/index.html.
8. "GitHub website." URL: https://github.com/.
9. "Google Code website." URL: http://code.google.com/.
10. "Google Play store website." URL: https://play.google.com/store.