International Journal of Multidisciplinary Approach and Studies

ISSN NO:: 2348 – 537X

Android Mobile Automation Framework

Pallavi Raut* & Satyaveer Tomar**

**Department of Computer Science & Engineering, RGPV, Bhopal,M.P*

*** Department of Computer Science & Engineering, SBITM, Betul, M.P*

ABSTRACT:

Earlier, mobile applications were offered for basic tasks such as email, contacts, calendar, music, weather, or stock information. Due to public demand and availability of advanced developer tools, mobile apps rapidly expanded into various categories such as games, factory automation, location-based services, banking, etc. Increasingly prevalent usage of mobile devices has raised the popularity of mobile apps; soon enough, functional testing of those apps has become an extremely important task.

Quality testing of mobile apps across various operating systems and devices is necessary for their long-term success in this highly competitive app market. There is a constant need for robust Android test automation tools to ensure software compatibility across various versions of the operating system. In case of Android, you have to also take care of the hardware diversity provided by many Android OEMs. Taking into account all of these issues and constraints, Android Mobile Automation Framework is developed to overcome these challenges. In this paper we present an approach for automating the testing process for Android applications, with a focus on GUI and functional bugs.

Keywords:- *Android SDK, ADB, Net beans, Eclipse-IDE, java/c++*

I. INTRODUCTION

Android is open-source software architecture provided by the Open Handset Alliance, a group of 71 technology and mobile companies whose objective is to provide mobile software platform.

The Android platform includes an operating system, Middleware and applications. As for the features, Android incorporates the common features found now a days in any mobile device platform, such as: application framework reusing, integrated browser, optimized graphics, media support, network technologies, etc.The Android architecture, depicted in Figure 1, is composed by five layers: Applications, Application Framework, Libraries, Android Runtime and finally the Linux kernel.

The uppermost layer, the Applications layer, provides the core set of applications that are commonly offered out of the box with any mobile device. The Application Framework layer provides the framework Application Programming Interfaces (APIs) used by the applications running on the uppermost layer. Besides the APIs, there is a set of services that enable the access to the Android's core Features such as graphical components, information exchange managers, event managers and activity managers, as examples. Below the Application Framework layer, there is another layer containing two important parts: Libraries and the Android Runtime. The libraries provide core features to the applications. Among all the libraries provided, the most important are libc, the standard C system library tuned for embedded Linux-based devices; the Media Libraries, which support playback and recording of several audio and video formats; Graphics Engines, Fonts, a lightweight relational database engine and 3D libraries based on OpenGL ES.

The important layer from test automation perspective is Application layer. Applications are the top layer in the android architecture and this is where applications are goanna fit.

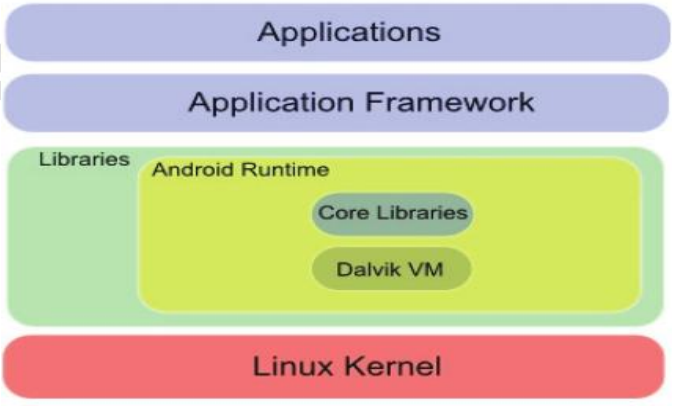


Figure 1: Architecture of Android platform.

Several standard applications comes pre-installed with every device, such as:

SMS client app

- Dialer
- Web browser
- Contact manager

Whereas third party developed application can be downloading either from google play or install it using command prompt if application file available.

2. RELATED WORK

2.1 Monkey Talk

Monkey Talk is a well-developed system that supports record, replay, and test automation across different technologies and frameworks including Android[7]. The system allows you to record and replay user inputs create automated user tests or run interactive tests through their IDE which is built on top of the popular eclipse IDE. Using Monkey Talk, one can connect to a virtual or physical device running Android and run their tests on it. From there, most of the user interactions can be recorded and are converted to their specific format including detailed information about the events that occurred and the elements they affected. Monkey Talk also provides a Java Script api which allows you to override event handlers to record custom messages.

2.2 Robotium

Robotium is an Android UI automation framework designed to make programmatic simulation of user actions on Android devices very simple. [8] It does not support any record or replay functionality as is but provides several mechanisms to ensure sanity in actions taken.

For example, when typing into a textbox or clicking one button it grants its user the ability to check that the desired elements exist and that their data or attributes are correct.

2.3 Deterministic Replay

A lot of research is being carried out in the area of UI testing for mobile apps, many of which involve record and replay. [1] Jason Flinn and Z. Morley Mao from the University of Michigan published a paper [1] about the applicability of deterministic replay for UI testing for mobile devices. Through their research they aimed at studying the challenges posed by implementing replay on phones. They also explored the benefits of replay, especially when it is performed remotely on cloud or cloudlet.

2.4 Guitar

GUITAR (Graphical User Interface Testing framework)is a test generation and automation framework that can be applied to GUIs of many kinds. [9] It has been extended to android applications as Android GUITAR. Android-Guitar is intended to simplify the testing process of GUIs on the Android platform by invoking GUITAR. A plug in is being developed that allows the GUITAR Ripper and Replayer to communicate with an Android application running on an Android emulator. This plugin is expected to facilitate automated and comprehensive testing of Android GUIs, as well as increase the breadth of GUITAR functionality.

3. CHALLENGES IN MOBILE WORLD

Testing mobile applications is more complex and time consuming compared to traditional desktop and web applications. The majority of desktop applications need to be tested on a single dominant platform – Windows. The lack of a similar dominant platform for mobile apps results in many apps being developed for and tested on Android, iOS and sometimes even more platforms. Challenges are

1. The biggest challenge when it comes to mobile application testing is the plethora of devices spread across different platforms. Obviously, it is not feasible to test application on

each and every available device which means you have to strategically choose a few physical devices.

One need to remember that testing on one device never assures it would work on any other device, irrespective of whether it is of same make, same OS Version or using the same platform! Not testing on a physical device always runs a risk of potential failure on that device, especially when the target audience for the application is widespread, like for a game.

Testing demands different physical devices to cover the following:

a) Varying screen sizes.

b) Different Form factors.

c) Different pixel density and resolution.

d) Different input methods like QWERTY, touch etc.

2. Different platform testing: In case of native app, it goes without saying that it will need dedicated testing effort on all platforms for which it is developed. It gets a bit tricky in case of HTML5 based hybrid applications. While the code remains same, lot of factors come into play on different platforms.

3. Testing on different OS versions of the same platform: Test your application on all major platforms aka Android, iOS, Windows etc but each one of them have several OS versions floating in market. An obvious choice is to test on the most recent versions of all the platforms but this would not do justice for Android application. The latest version of Android is Jellybean introduced quite a while ago, still there are lot of devices which have not yet received OS updates (and possibly will never be updated). Its interesting to note a big difference in Google's and Apples's approach in handling the OS updates. While the former relies on device manufacturers to update the respective devices, Apple handles the updates itself resulting into mass updating of all Apple devices as soon as a new OS version is released. Whatever is the OS version on a device, user can still install your application and use it, which calls for testing different OS versions.

4. Testing on various networks and network vendors: Most of the mobile applications require network connectivity sometime or the other. If the app talks to a server for flow of information to and fro, testing on various (at least all major ones) networks is important. Mobile networks use different technologies like CDMA and GSM with their 2G, 3G and 4G versions. The network infrastructure used by network operators may affect data communication between app and the backend. Apart from the different operators, an application needs to be tested on Wi-Fi network as well.

5. Mobile environment: It poses another unique challenge to the tester. Mobile environment is very dynamic and has constraints like limited computing resources or available memory and battery life

4. PROPOSED SCHEME

This Android Mobile Automation Framework is based on robotium. This is an open source Android testing framework with robust functionalities to cover almost all possible scenarios encountered in android applications. It has powerful features which make this framework for android Black-box testing to develop test scripts for functional, system as well as acceptance test scenarios.

When it comes to testing mobile devices, there are two fundamental ways to approach the testing process. The first way is to use an emulator, which is a software application that allows you to reasonably simulate the behaviour of a mobile application on a given mobile device configured in a certain way. While emulators are quite useful, they are not to be relied upon solely due to limitations in the emulation software. The second way is to use the actual devices you are targeting in the mobile marketplace. The test cases written using this framework can either be executed on the Android Emulator Android Virtual Device (AVD)) or on a real Android device.

Below architecture depicts the Android mobile automation framework.

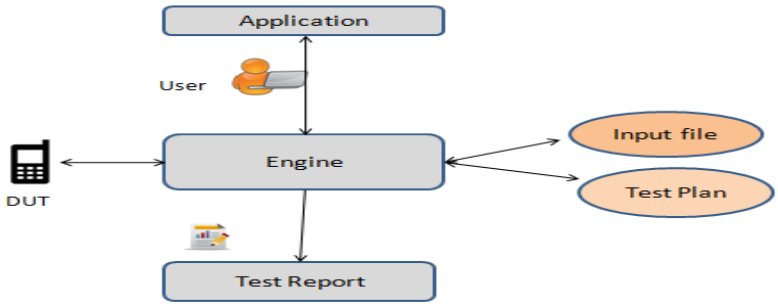


Fig2: Architecture of AMAF

Xml file is one of the input files of the Android Mobile Automation framework (AMAF). It contains the steps required to navigate through the various screens in an application. A typical xml file contains the name of the application as the entry point (parent tag) and the derived commands as the child tags. When test script is developed, it will be compiled using Eclipse and then will upload the app to device with help of framework which then invoke the test script on device and start execution

Fig: 3 AMAF Home screen

5. IMPLEMENTATION AND RESULTS

5.1 Test Case Generation

Eclipse is a development environment that has been extended by AMAF with the necessary functionality to create test scripts against mobile applications. The benefit of using Eclipse for creating automated test cases is that you have now one platform for development and debugging, scripts can run in parallel on different mobile devices and compiled test scripts.

AMAF test can access the attributes of the user interface elements as they are defined in the mobile operating system. This is an essential technique that has been used by test automation tools on the PC for many years.

AMAF is a testing framework for java applications, integrated in the Android development environment. JUnit can generate several classes of test cases based on the application source code. Since activities are the main entry points and control drivers in Android applications, test case generation is based on activities. We first identify all activities in an application and then use the Activity Testing class in Junit to generate test cases for each activity. Test script will be generated and placed at appropriate folder in AMAF framework.

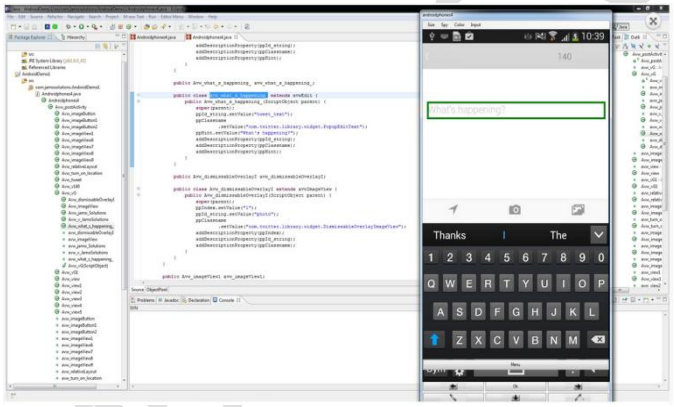
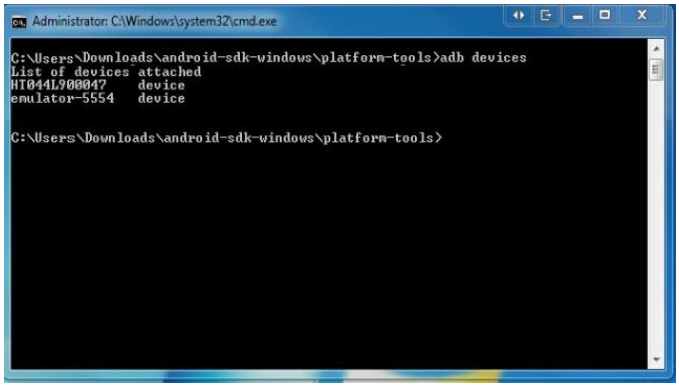


Fig: 4 Test case generation

5.2 Test Execution Environment

Once the test cases developed, In first panel, select the test scripts that needs to be execute and then create the test suite.

In order to display device in device list, adb path needs to be set up in environment variable and then only in device details panel, select the device details and test suite that created in first panel. Device connected to AMAF should display like below in fire of adb command



After that select the device and test suite in execution panel and start execution. Script will run on device.

5.3 Test Case Results Analysis

After execution of all test script, test result summary will be generated. Test results can be exported in HTML or CSV format. This test result summary contents result of test script Passed or Failed. Option to view either passed or failed test cases is available.

All | Pass | Fail

TestCase_ID	TestCase_Description	Result	Reason
TC_01	Verify the landing page of portal application	Pass	
TC_02	Validate the content on the Benefits for portal application	Pass	
TC_03	Validates UI of Home screen of "Happening now" screen	Pass	
TC_04	Validates Share your Funny page success Message after valid information submission	Pass	
TC_01	Tests all Pages	Pass	
TC_02	Validates contact us page success Message after valid information submission	Pass	
TC_03	Validates Channel Finder Page	Pass	
TC_04	Validates Share your Funny page success Message after valid information submission	Pass	

6. CONCLUSION

In this paper, a technique for automatic testing of Android mobile applications has been proposed. The technique is based on robotium and is used to develop test cases that reveal application faults like run-time crashes, or that can be used in regression testing. Test cases consist of event sequences that can be fired on the application user interface. At the moment, we have not considered other types of events that may solicit a mobile application (such as external events produced by hardware sensors, chips, network, or other applications running on the same mobile device) and just focused on user events produced through the GUI.

The proposed testing technique aims at finding GUI, functional and user-visible faults on modified Versions of the application. This framework will be worked on both Emulator and physical android device.

Benefits of this automation framework are

- Framework has capacity to handle multiple activity
- One script will run on all android platform versions
- 43% of efforts save per cycle compared to manual testing as shown in table 1
- Based on Junit, opening the door for Unit Testing with Android
- Maintenance of the script is very easy
- Support Native as well as Hybrid application

Total Effort Estimation				
Testing Cycle	Test cases	Total Efforts (PD)	Person Weeks(PW)	Automation Efforts(PD)
First Cycle	1000	10	2	5
Second Cycle	1000	10	2.0	5
Regression Release	1200	8	1.6	5.625
Total Effort		28	6	16
Total Effort (PM)		1.33		0.76
			% Efforts Saved	43%

Table 1 Automation Results

REFERENCES:

i. DomenicoAmalfitano, Anna Rita Fasolino, Porfirio Tramontana "A GUI Crawling-based technique for Android Mobile Application Testing"DOI: 10.1109/ICSTW.2011.77 Conference: Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference

ii. Cuixiong Hu&IulianNeamtiu "Automating GUI Testing for Android Applications" Department of Computer Science and Engineering, University of California, Riverside, CA, USA http://www.cs.ucr.edu/~neamtiu/pubs/ast11hu.pdf

iii. Gerrard Paul, "Testing GUI Applications", EuroSTAR Conference, Edinburgh, November 1997, http://www.gerrardconsulting.com/GUI/TestGui.html, 18.07.2010

iv. Garima Pandey, DikshaDani" Android Mobile Application Build on Eclipse" International Journal of Scientific and Research Publications, Volume 4, Issue 2, February 2014, ISSN 2250-3153

v. Testing for poor responsiveness in android applications ShengqianYang ;Dacong Yan ; Rountev, A. Engineering of Mobile-Enabled Systems (MOBS), 2013 1st International Workshop on the DOI: 10.1109/MOBS.2013.6614215 Publication Year: 2013

vi. Using GUI ripping for automated testing of Android applications Amalfitano, D. ;Fasolino, A.R. ; Tramontana, P. ; De Carmine, S. ; Memon, A.M.Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on DOI: 10.1145/2351676.2351717 Publication Year: 2012

vii. Gorilla Logic, MonkeyTalk. https://www.gorillalogic.com/monkeytalk

viii. Renas Reda, Robotium - The World's Leading Android Test Automation Framework, User scenario Testing for Android. https://code.google.com/p/robotium/.

ix. Nguyen, Bao, Bryan Robbins, and Ishan Banerjee, GUITAR - A GUI Testing Framework Event Driven Software Lab - University of Maryland. http://sourceforge.net/projects/guitar/

x. Shyam Bhati, Sandeep Sharma, Karan Singh "Review On Google Android a Mobile Platform" IOSR Journal of Computer Engineering(IOSR-JCE) e-ISSN: 2278-0661, p-ISSN: 2278-8727Volume 10, Issue 5 (Mar. - Apr. 2013), PP 21-25

xi. Khawlah A. AI-Rayes, AiseZulalSevkli, Hebah F. AI-Moaiqel, Haifa M. AI-Ajlan, Khawlah M. AI-Salem, Norah I. AI-Fantoukh "A Mobile Tourist Guide for Trip Planning" IEEE MULTIDISCIPLINARY ENGINEERING EDUCATION MAGAZINE, VOL. 6, NO. 4, DECEMBER 2011

xii. R. Gove and J. Faytong. Identifying infeasible GUI test cases using support vector machines and induced grammars. In TESTBED, pages 202–211, 2011.

xiii. F. Gross, G. Fraser and A. Zeller. Search-based system testing: High coverage, no false alarms. In ISSTA, pages 67–77, 2012.

xiv. S. Hao, D. Li, W. G. J. Halfond, and R. Govindan. Estimating mobile application energy consumption using program analysis. In ICSE, 2013.

xv. C. Hu and I. Neamtiu.Automating GUI testing for Android applications. In AST, pages 77–83, 2011.

xvi. J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, and T. Millstein. Dr. Android and Mr. Hide: Fine-grained permissions in Android applications. In SPSM, 2012.

xvii. M. Jovic, A. Adamoli, and M. Hauswirth. Catch me if you can: Performance bug detection in the wild. In OOPSLA, pages 155–170,2011.

xviii. M. Jovic and M. Hauswirth. Listener latency profiling: Measuring the perceptible performance of interactive Java applications. Science of Computer Programming, 76(11):1054–1072, 2011