International Journal of Current Engineering and Technology
E-ISSN 2277 – 4106, P-ISSN 2347 - 5161

Available at http://inpressco.com/category/ijcet

Research Article

Software Testing Automation using Appium

Gaurang Shah[Å*], Prayag Shah[Å] and Rishikesh Muchhala[Å]

[Å]Information Technology Department, DJSCOE, Vile Parle (W), Mumbai -400056, India

Accepted 20 Sept 2014, Available online 01 Oct 2014, **Vol.4, No.5 (Oct 2014)**

Abstract

Software testing *is an important, costly and one of the most tedious processes in* ***software development life cycle****.* ***Automation*** *of software testing is a boon for companies who develop applications on a large scale. This paper is produced so as to put some light on the latest* ***automated software testing technologies*** *and mainly talks about a testing tool called* ***Appium****. The main aim of automating the software testing process is to produce a high quality, optimized and a complete software and deliver it to the customer in the shortest possible time.*

Keywords: *software testing, automation, software development, Appium, mobile applications, software engineering*

1. Introduction

Software testing phase within the software development lifecycle is the phase which ultimately determines how sellable our product is. It is in this sense that a lot is being explored in terms of optimizing testing. With the advent and ever increasing use of mobile technology, testing also has to keep up with the complexities in terms of both volume and variety. The volume is in terms of the number of test cases involved while testing. The variety is in terms of defining the test cases such that it accommodates the nuances of touch screen, multitasking, mobile internet and other gesture recognitions. Automation is the solution to deal with volume and ironically automation also is the challenge to deal with variety. In this paper, we try to emphasize on the need for automating testing for mobile technology. We also highlight the challenges involved for doing the same, given the overwhelming heterogeneity within the application/product. We also discuss an open source tool for software automation called Appium as an example which will give us an insight to providing solutions to the above mentioned challenges.

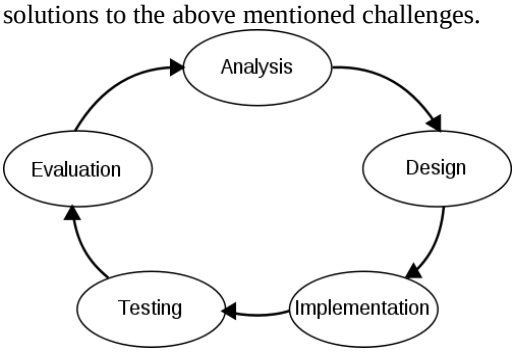

Fig.1 Software Development Life Cycle

Learning from the tools available for automating testing for desktops, tools like MonkeyTalk, KIF, Calabash, etc. were developed for mobile native applications. Most of these tools required an extra agent that needed to be compiled along with the application code. The extra agent was needed so that the tool can interact with the mobile application. The libraries of this extra agent had to be removed at the time of submitting the application to the Store. Appium, on the other hand, does not need any such extra agents to be included in the original code. Appium can be termed as a revolutionary tool that can completely change the testing process in a much efficient and swift way.

Appium has improved significantly since its inception and is constantly being added up with new features. Although the version for MacOS and Windows differs in some small manners but is overall very similar for both the operating systems. Appium has three main components viz. Appium server, Inspector and Doctor. These components will be highlighted in the components section of this paper.

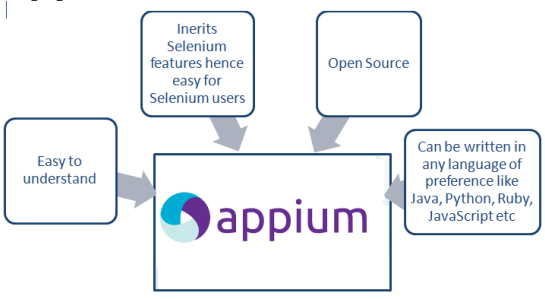

Fig.2 Block Diagram of Appium

2. Concepts

2.1 Client/Server Architecture

Appium is a webserver that exposes a REST API. It receives connections from a client, listens for commands, executes those commands on a mobile device, and

*Corresponding author: **Gaurang Shah**

responds with an HTTP response representing the result of the command execution.

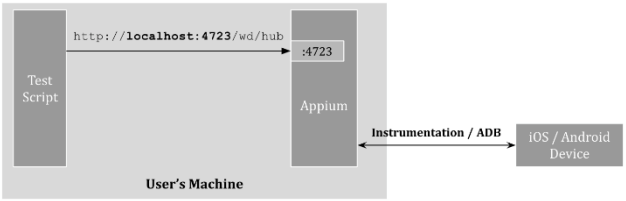

Fig.3 Appium client/server architecture

The fact that we have a client/server architecture opens up a lot of possibilities: we can write our test code in any language that has an http client API, but it is easier to use one of the Appium client libraries. We can put the server on a different machine than our tests are running on.

2.2 Session

Automation is always performed in the context of a session. The session takes place in following steps:

- Clients initiate a session with a server and it sends a JSON object called the 'desired capabilities' object.
- At this point the server will start up the automation session and respond with a session ID which is used for sending further commands.

2.3 Desired Capabilities

Desired capabilities are a set of keys and values (i.e., a map or hash) to notify the Appium server about the kind of automation session we want to start up. Various capabilities can be defined using which we can change the behavior of the server according to the requirements. For example, we might set the '**platformName**' capability to '**iOS**' to notify to Appium that an **iOS** session is to be started up, rather than an Android one.

2.4 Appium Server

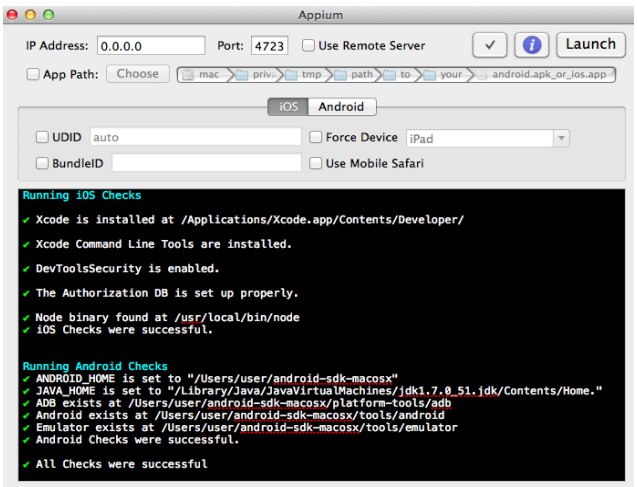

Fig.4 Appium server

Appium is a server written in Node.js. It can be built and installed from source or installed directly from NPM. We can modify capabilities of the server according to the test requirements. Appium server handles the execution of script and linking it with the simulator/emulator. Fig.4 shows the Appium server.

2.5 Appium Clients

Client libraries are available in Java, Ruby, Python, PHP, JavaScript and C#. All of these support Appium's extensions to the WebDriver protocol. When using Appium, we have to use these client libraries instead of the regular WebDriver client.

2.6 Appium.app, Appium.exe

GUI wrappers around the Appium server exist and can be downloaded. These come bundled with everything required to run the Appium server. These wrappers also come with an Inspector, which enables you to check out the hierarchy of the application. Inspector simplifies the task of writing various tests and comes in handy when there are lot of elements present in a window.

3. Architecture

Appium is an HTTP server written in node.js which creates and handles multiple WebDriver sessions for different platforms like iOS and Android for native, web as well as hybrid applications.

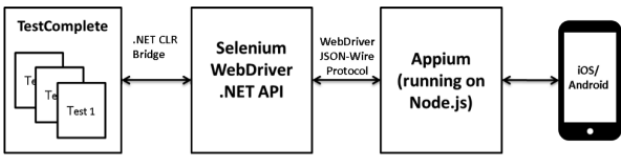

Fig.5 Basic architecture of Appium.

Appium starts a test case using a script on the device that listens for proxy commands from the main Appium server. Each vendor like iOS and Android have a different way and mechanism to run a test case on the device so Appium kind of hacks in to it and runs this test case after listening commands from Appium server.

4. Components

This section discusses the components of Appium tool in depth. There are three major components in Appium viz. Inspector, Doctor and Appium Server. The Appium server is already described in sub section *2.4.* The other two are described below.

4.1 Inspector

Appium provides an inspector that helps the test engineer to detect, inspect and interact with user interface elements. Inspector provides the test engineer with the path of every single element present of the current screen on simulator

or a real device. Inspector has a record function which helps in generating scripts automatically upon interaction with UI elements. Inspector provides different ways of interacting with the GUI of the application under test. Given below is a figure showing the Appium inspector.

Fig.6 Appium inspector.

4.2 Doctor

Appium provides a doctor for a very simple task i.e. to run a check whether all the components required for the running of test and simulators are met beforehand. It shows an error if one or more components required are missing and without that the test cannot be run. The results of the doctor's inspection of requirements is shown on the Appium server window. If all conditions are met, there is a green tick in front of each requirement.

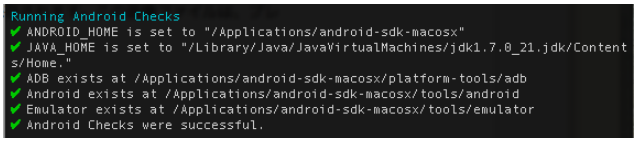

Fig.7 Results of a check by the doctor.

5. Working

5.1 Working of Appium in iOS

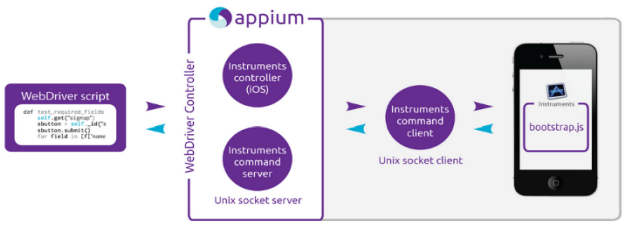

Fig. 8 Appium working in iOS

On iOS, Appium proxies command to a UIAutomation script running in Mac Instruments environment. Apple provides this application called 'instruments' which is used to do lot activities like profiling, controlling and building iOS apps but it also has an automation component where we can write some commands in JavaScript which uses UIAutomation APIs to interact with the App UI. Appium utilizes these same libraries to automate iOS Apps.

In the above figure, we can see the architecture of the Appium in context to the iOS automation. A command life-cycle goes like, Selenium webdriver picks a command from the code like (Element.click) and sends it in form of JSON via http request to the Appium server. Appium server knows the automation context like the iOS and Android and sends this command to the Instruments command server which will wait for the Instruments command client (written in node.js) to pick it up and execute it in bootstrap.js with in the iOS instruments environment. Once the command is executed the command client sends back the message to the Appium server which logs everything related to the command in its console. This cycle keeps going till the time all the commands gets executed.

5.2 Working of Appium in Android

The situation is almost similar in case of Android where Appium proxies commands to a UIAutomator test case running on the device. UIAutomator is Android's native UI automation framework which supports running jUnit test cases directly in to the device from the command line. It uses java as a programming language but Appium will make it run from any of the WebDriver supported languages.

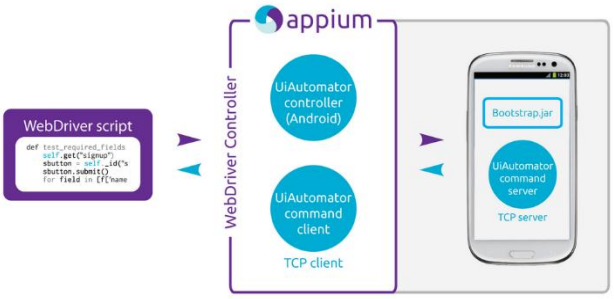

Fig.9 Appium working in Android

In the above diagram we can see, here we have Bootstrap.jar in place of bootstrap.js which represents out test case when compiled in java. As soon as it gets launched it spawns a TCP server. Here the TCP server resides inside the device and client is in the Appium process which is just opposite to the way it is in iOS.

6. Remarks

6.1 Advantages

\

- The experience of the developer remains same irrespective of the platform he/she is working on.
- Appium supports multiple scripting languages which means developers having diverse expertise can use the same tool.
- At any point of time, the Appium testing tool does not make any changes in the original code of the application. Which means we submit the same code we test.

- Appium provides cross-platform mobile testing which means the same tests hold true while working on different platforms.
- As Appium is open source, there is a complete user support for filing bugs on github.

6.2 Disadvantages

- There is a technical limitation i.e. on iOS we can only run one instance of simulator on instruments per Mac OS. This means that we can only run our iOS scripts on one device per Mac machine.
- If we want to run our tests on multiple iOS devices at the same time then we would need to arrange the same number of Mac machines, which would be costly affair. But this limitation can be resolved if we execute our scripts in Sauce Lab's mobile cloud which at present supports running scripts on multiple iOS simulators at the same time.
- Appium uses UIAutomator for Android automation which only supports Android SDK Platform, API 16 or higher. To support the older APIs, another open source library called Selendroid is used.

Conclusion

Continuous advancements in mobile applications is taking place and today we need high performance applications designed, developed and developed as quickly as possible. Testing is the most important step before launching such applications especially when developed for use in critical areas of the market where a small error can lead to a huge failure. Thus, automation of software testing is the new trend taken up by developers to ensure a high performance application in a short period. Appium seems to be much more promising in this aspect as it delivers powerful features to test engineers which can save a lot of time, labor and cost of the project. Thus, Appium provides a complete new revolution in automation testing which promises efficient, bug-free and quality-rich applications.

Acknowledgement

We would like to thank our honorable principal Dr. Hari Vasudevan of D. J. Sanghvi College of Engineering and Dr. A. R. Joshi, Head of Department of Information Technology, for giving us the facilities and providing us with a propitious environment for working in college. We would also like to thank S.V.K.M. for encouraging us in such co-curricular activities.

References

Rankin C. (2002), The Software Testing Automation Framework, *IBM Systems Journal*, Vol. No. 41, Issue: 1

Gao, J. Xiaoying Bai ; Wei-Tek Tsai ; Uehara, T. (2014), Mobile Application Testing: A Tutorial, Computer, Vol. No.47, Issue: 2.

Appium
>http://en.wikipedia.org/wiki/Appium

Test automation
>http://en.wikipedia. org/ wiki/ Test_ automationAppium

API reference
>http:// appium.io/slate/en/master/?java#