Automation Testing with Appium Framework in IP Multimedia Subsystem

Hieu Minh Tran
Viettel High Tech
Viettel Group
Hanoi, Vietnam
hieutm15@viettel.com.vn

Thinh Duc Tran
Viettel High Tech
Viettel Group
Hanoi, Vietnam
thinhtd4@viettel.com.vn

Linh Duc Nguyen
Viettel High Tech
Viettel Group
Hanoi, Vietnam
linhnd23@viettel.com.vn
linhnd.ac@gmail.com

Tuan Duc Ninh
Viettel High Tech
Viettel Group
Hanoi, Vietnam
tuannd43@viettel.com.vn
ninhductuanit@gmail.com

Vuong Van Ngo
Viettel High Tech
Viettel Group
Hanoi, Vietnam
vuongnv61@viettel.com.vn
vuongvuongnvvt@gmail.com

***Abstract*— The IP Multimedia Subsystem (IMS) refers to the standard for a telecommunication system that controls multimedia services accessing different networks. Not only the IMS but other telecommunications networks are also required to operate continuously. In addition, any system error can lead to massive losses in revenue and productivity. Therefore, testing and quality control task has an important role in telecommunications networks, and it requires an efficient method to detect any inherent errors and failures. To facilitate the testing process, a solution using the test automation tool Appium is proposed. The Appium framework is mainly used to validate mobile browsers and mobile applications with UI. If the IMS can be implemented with the Appium framework, the burdensome testing tasks can be eased. Therefore, we developed an automated model in which the Appium framework can help in testing functions of the IMS. Our model contains some new features added to the basic function of the framework and these features are designed and developed specially for testing the IMS services. This proposed solution not only reduces the time and labor in testing but also can be applied to other systems that are related to mobile phone services.**

Keywords—IMS, Appium, testing, manual, automation, mobile phone, telecommunications

I. Introduction

A. IP Multimedia Subsystem

The IP Multimedia Subsystem (IMS) is the key element in the 4G/5G architecture that allows real-time services on top of the Universal Mobile Telecommunications System (UMTS) packet-switched domain [1][2]. IMS is based on the specification of Session Initial Protocol (SIP), Diameter, H.248 protocols as standardized by the Internet Engineering Task Force (IETF) [3][4][5]. These protocols' messages are transmitted over UDP/TCP. The IMS system with main protocols is illustrated in Fig. 1. The IMS provides many services related to communications, for instance:

- HD Voice Call
- Video and text messaging over IP networks
- Telephone call control: call barring, call forwarding, conference call…

To validate all services of the IMS, a suite of test cases is developed. Then the IMS must pass this test cases suite. The model of testing is shown in Fig. 2. The development of testing methods is described in the next sections.

B. Brief Introduction of Testing Methods

There are some methods of testing: manual testing and automation testing. Each method has its advantages and drawbacks [6].

1) Manual testing

Manual testing is a type of software testing in which testers execute test cases manually without using any automated tools. Manual testing is the most fundamental technique of all testing types as it can detect many kinds of defects and help to find critical errors. Any new services must be manually tested before their testing can be automated. Therefore, manual testing is indispensable.

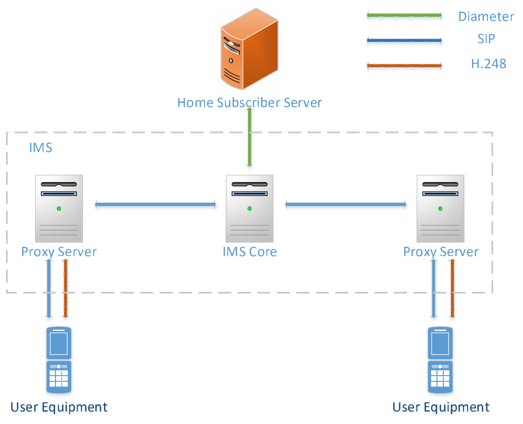

Fig. 1. IMS model with main protocols

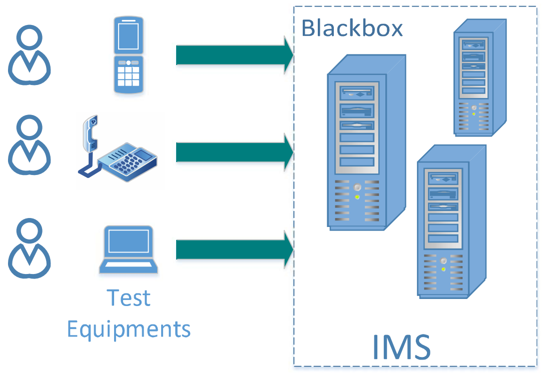

Fig. 2. IMS testing model

2) *Automation testing*

Most systems are set up initially with manual testing. Testers can ensure that the test cases cover all the functions of the systems and their operations are flawless. However, the number of test cases gradually increases as more and more functions are developed. At this time, another type of testing is required.

Automation testing refers to a type of testing in which developers or testers write the test script once with the help of testing tools and frameworks and run it on the applications. The script automatically performs test cases without human intervention and reports the result (the number of errors, the time of errors, and the total testing time…). Automation testing generates faster and more accurate results compared to manual testing. The comparison of automation testing and manual testing is shown in Tab. I.

TABLE I. COMPARISION BETWEEN AUTOMATION TESTING AND MANUAL TESTING

Manual testing	Automation testing
Involving human resources	Using automation tools
Time-consuming	Faster testing
Repetitive and error-prone	Accurate
Suitable for small test cases suite	Efficient with great test cases suite

II. RELATED WORKS

First, the IMS system was tested by Testing and Test Control Notation version 3 (TTCN-3) [7][8]. TTCN-3 was standardized by the International Telecommunication Union Telecommunication Standardization Sector (ITU-T). The flow of mobile phone calls is simulated on a TTCN-3 sever. TTCN-3 test cases can be configured to perform automatically and the TTCN-3 server can cover a great number of test cases. However, sometimes the simulation is not similar to the operation of mobile phones. For example, some phones reject unsuitable messages but TTCN-3 test cases cannot detect that kind of messages in advance. Besides, the firmware of mobile phones is updated frequently, then testers must find which test cases are required to be rewritten.

Therefore, the IMS system needs another automation testing tool. Nowadays, the Appium framework is used widely in automation testing and it is mainly used to test mobile applications. Besides, there are some other automation testing tools, namely Selenium, Monkey Talk, and Robotium. However, the Appium framework has some advantages that make it suitable for the IMS. First, the Appium framework is an open-source platform. Second, the Appium framework supports both iOS and Android platforms. On the other hand, Monkey Talk and Robotium are not open-source platforms and Robotium only supports the Android platform. Compared to other automation tools, the Appium framework stands out as a flexible framework that supports various programming languages and mobile platforms.

III. APPIUM FRAMEWORK

A. *Introduction*

Appium is an automated mobile testing tool, which is used to test applications [9][10]. The model of the Appium framework consists of three components:

- Appium Client
- Appium Server
- End devices

The architecture of the Appium framework is depicted in Fig. 3. End devices refer to some physical devices connected to the Appium Server.

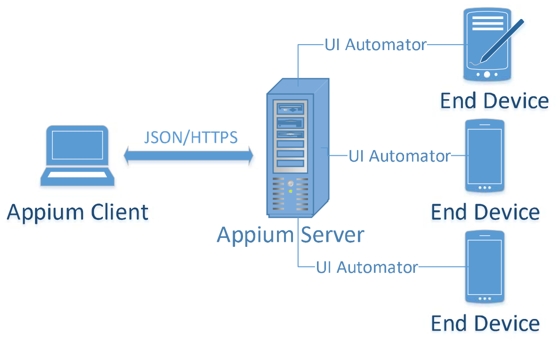

Fig. 3. Architecture of Appium framework

B. *Appium Client*

Appium Client is a scripted code that is written in any programming language like Java, Python, etc. This script holds the configuration details of the end devices and the applications. The code to run the test cases of the applications is also scripted here.

C. *Appium Server*

Appium Server receives requests from the Appium Client in JSON format and executes those commands on the test devices. It creates sessions to interact with the devices and then forwards the requests from the client to the devices. The UI Automator is a framework that provides a set of APIs to build user interfaces. These interfaces allow users to perform operations such as opening the settings menu or the app launcher on the end devices.

D. *Testing flow with the Appium framework*

The steps to test with the Appium framework are shown in Fig. 4. These steps are described as follows:

- **Step 1**: Testers prepare scripts containing the Appium Server configurations and test cases.
- **Step 2**: The scripts are converted into JSON format. Then the Appium Client sends commands in JSON format to the Appium Server.
- **Step 3**: The Appium Server recognizes the commands and establishes connections with the corresponding end devices. Once connections are made, it triggers the execution of test cases in the end devices.
- **Step 4**: The end devices respond to the Appium Server and the responses are in the form of HTTP. The Appium Server verifies the responses to determine whether the test cases fail or pass.
- **Step 5**: The Appium Server generates the result of the test process. It also provides the log of all actions performed in the end devices.

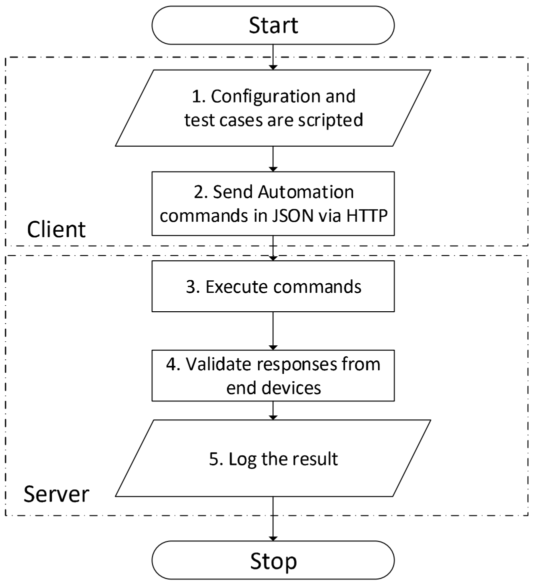
Start
1. Configuration and test cases are scripted
2. Send Automation commands in JSON via HTTP
Client
3. Execute commands
4. Validate responses from end devices
5. Log the result
Server
Stop

Fig. 4. Appium test flow

IV. Implement Appium Framework in IMS

In this section, the procedure of implementing the Appium framework in the IMS is detailed.

A. Model

Fig. 5 depicts how the Appium framework is implemented in the IMS. Test devices, the Appium Server, and the Appium Client are set up before the test process. The script in Appium client consists of each step to conduct the test process. For each test case, the Appium Client sends requests to the Appium Server and then the Server controls test devices to perform the test case. For example, if the test case is testing call barring, the steps are as follows:

- **Step 1**: The client starts the call barring test case and sends requests to the server.
- **Step 2**: The server orders the test devices to register call barring service.
- **Step 3**: The test devices automatically send text messages or send USSD code to the telecommunications core for registering call barring.
- **Step 4**: The test devices wait for responses from the telephony center to check whether the call barring registration is successful or not.
- **Step 5**: If the call barring registration is successful, the test devices make some calls to each other. If the calls are connected normally, this test case is considered failed. If there is an announcement about barring service and the test devices cannot make the calls, this test case is considered a validated test case.
- **Step 6**: The server requests the test devices unregister all the call services (including call barring).

B. Issues

The Appium framework is firstly designed for UI testing and it is mainly used for mobile app testing. For testing services of the IMS systems, some issues must be measured and they are described as follows:

a) Telecommunications systems consist of many subsystems and defects can occur at any subsystems. When a test case fails, the failure may not come from the IMS but from other neighbor subsystems. For example, when wave signal receiving at test devices is weak, all the devices' calls and data services are dropped or not stable. This problem belongs to the handshake procedure between mobile phones and the access network.

b) A test device or the Appium Server malfunction then affect test cases' result.

c) The IMS is a telecommunications system that contains many applications. Moreover, these applications connect with each other as well as other network applications via TCP/UDP links. Therefore, there is a chance that the test devices cannot receive expected messages or the expected messages arrive late.

d) Web testing or application testing with the Appium framework usually requires only one end device. On the other hand, test cases of call sessions involve many end devices. The Appium framework supports recording screens of end devices when their test case is performed. However, many devices correspond to many videos, and testers find it hard to watch many videos simultaneously.

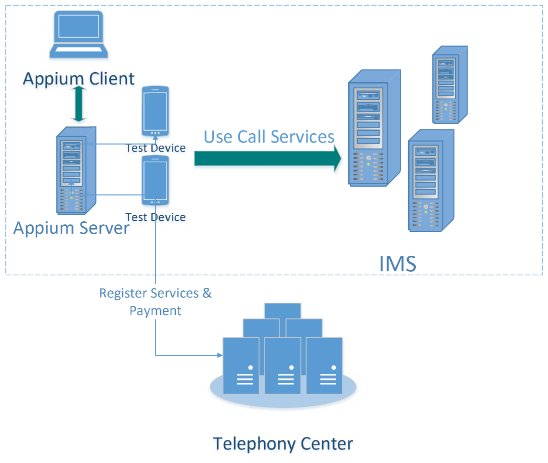

Fig. 5. Model of the Appium framework implementation

C. New Features

To overcome these issues, some new features are developed:

a) **Test cases retrying function**: This function is developed to collect all failed test cases that are reported by the Appium Server. Then it requests the Appium Server to perform all these failed test cases again. This solution can solve the problems related to the neighbor subsystems of IMS. Telecommunications systems are designed as fault-tolerant systems, so an error seldom occurs repeatedly. Therefore, at the second attempt, the failed test cases can be conducted successfully.

b) **Devices reboot function**: When a test device performs many failed test cases, this device may break down. The reboot function requires that this device must reboot. After the device reboots successfully, the Appium Server continues the test process. Failed test cases can be performed again with the retrying function. Besides, the Appium Server can malfunction and report some errors. The reboot function checks the logs of failed test cases. Then it looks up for the Appium Server errors keywords, for instance, "socket hangup". The function will reboot all devices and the Appium Server, then make new connections if there are errors related to the Appium Server.

c) **Services validation function**: When a test device registers or unregisters any call services, it must wait for the confirmation text messages from telephony centers. If these text messages arrive late or are discarded somewhere, the test device needs to check whether its services are registered or unregistered successfully. The device can send a USSD code to validate its services. It also can register or unregister services again to ensure the script is performed correctly.

d) **Monitoring function**: The videos of all devices in one test case are merged into one video. Testers can watch the screen recordings of all end devices in one monitor screen and they can see how the devices interact with each other.

D. Advantages of Automation Testing using the Appium framework in IMS

When the Appium framework is implemented in the IMS for Automation testing, it brings many benefits. The advantages of automation testing with the Appium framework in the IMS are listed as Management Graphical User Interface (GUI), Object Oriented Programming (OOP) design, Time-saving testing, and labor-saving testing.

1) Management Graphical User Interface

This GUI is developed to facilitate the automation testing of the IMS. All operations in test devices are implemented in GUI. The functions which are described in Section IV are also integrated into the GUI. Users can reboot devices, and check their prepaid balance… by pushing some buttons on the GUI. Moreover, GUI allows users to configure some parameters of test cases, the number of test devices, and which suite of test cases is tested. The GUI is shown in Fig. 6. After a test process finishes, the result is exported in Excel which is a user-friendly format.

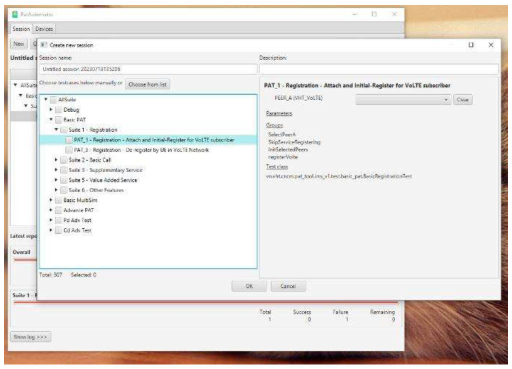

Fig. 6. Management GUI

2) Object Oriented Programming design

OOP design facilitates the process of creating new test cases. The OOP functions which interact with end devices can also be reused and developed without effort.

3) Time-saving and labor-saving testing

Table. II shows a comparison between manual testing and automation testing with the same suite of 600 test cases. Executing repeated tasks is tedious and testers can make mistakes in this process. The Appium framework eliminates inherent errors related to the tester's operations.

TABLE II. AUTOMATION TESTING AND MANUAL TESTING RESULT

	Manual testing	**Automation testing**
Total time	80 hours	16 hours
Operation time	8 hours/day	24 hours/day
Labor requirement	Require testers	Not require any testers
Report	Test reports are written by testers manually	Test reports are exported by the Appium framework

V. PRACTICAL APPLICATION

The implementation of automation testing with the Appium framework has been applied to the IMS system of Viettel High Tech, Viettel Group in Vietnam. Automation testing greatly reduces the time required for launching new services in the IMS. The system has operated in stable condition and serviced more than 10 million users for 2 years.

VI. CONCLUSION

At first, the IMS is assessed by manual testing. However, when the number of services in the IMS increases, the number of test cases increases as well. Therefore, another solution for testing is required. Then automation testing with the Appium framework is chosen. Before being implemented into the IMS, the Appium framework needs some new features to adapt for telecommunications testing. With the development of these new features, our IMS model with the Appium framework is deployed successfully. This helps to reduce the time and labor needed for testing. Our proposed solution can be applied to other telecommunications networks which involve telephony devices.

REFERENCES

[1] 3GPP TS 23.228, "IP Multimedia Subsystem (IMS); Stage 2,".

[2] 3GPP TS 23.002, "Universal Mobile Telecommunications System (UMTS; LTE; Network architecture,".

[3] RFC 3261, "SIP: Session Initiation Protocol".

[4] RFC 6733, "Diameter Base Protocol".

[5] RFC 3525, "Gateway Control Protocol Version 1".

[6] B. Kumari, N. Chauhan, Vedpal, "A Comparison between Manual Testing and Automated Testing," in Journal of Emerging Technologies and Innovative Research, vol. 5, issue 12 2018, pp. 323–331.

[7] ETSI ES 201 873-1 v4.7.1, "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", 2016-06.

[8] T. Vassiliou-Gioles, "A simple, lightweight framework for testing RESTful services with TTCN-3", in IEEE 20th International Conference on Software Quality, Reliability and Security Companion, 2020, Macau, China.

[9] J. Wang, J. Wu, "Research on Mobile Application Automation Testing Technology Based on Appium", in International Conference on Virtual Reality and Intelligent Systems, 2019, Jishou, China.

[10] A. M. Sinaga, P.A. Wibowo, A. Silalahi, N. Yolanda, "Performance of Automation Testing Tools for Android Applications", in 10th International Conference on Information Technology and Electrical Engineering, 2018, Bali, Indonesia.