Indian Journal of Science and Technology, *Vol 8(15), DOI: 10.17485/ijst/2015/v8i15/53253, July 2015*
*ISSN (Print) : 0974-6846*
*ISSN (Online) : 0974-5645*

# Service based Mobile Test Automation Framework for Automotive HMI

**Rajkumar J. Bhojan[1*], K. Vivekanandan[2], Subramanian Ganesan[3] and Pankaj Moses Monickaraj[4]**

[1]Wipro Technologies, Bangalore - 560100, India; jbrkumar@gmail.com
[2]BSMED, Bharathiar University, Coimbatore – 641046, India; vivekbsmed@gmail.com
[3]Electrical & Computer Engineering, Oakland University, Rochester, MI, USA; ganesan@oakland.edu
[4]Christ University, Bangalore - 560029, India; pankj.moses@gmail.com

## Abstract

**Objectives:** To implement a Service based Test Automation framework for testing web services in automotive Human Machine Interface (HMI) application. In this paper, we propose a service based test automation framework for web service testing in automotive HMI (Human Machine Interface). **Methods:** There are number of testing frameworks available for testing mobile application; In this proposal, an automated testing framework has been designed to test services which communicate between mobile application and real vehicles (cars) using differenttechnologies web services and mobility testing tools. **Findings:** In our findings, a customized mobile test automation framework has been developed which can eliminate most of the errors that may exist in manual testing by human and reduce a lot of time. Moreover, the framework can easily be extended to find a broader range of testing in automotive mobile applications. The techniques and mechanisms used in creating a test automation framework leads to analyze how they can do regression testing on mobile services in an efficient manner.

**Keywords:** Automotive Testing, Human Machine Interface, Mobile Test Automation, Regression Testing, Web Services

## 1. Introduction

Presently, web services are having an increasingly important role in the construction of computer applications used remotely by many users. Web services are Web components only accessible via interfaces published in standard Web services-specific interface definition languages (e.g. WSDL) and accessible via standard network protocols (e.g. SOAP). Therefore, how to design test cases for a Web service using its limited information exposed remains a challenge. Second, one major goal of adopting the Web services technology is to dynamically compose existing Web services as components to quickly construct a new service. Therefore, Web services testing usually imply that multiple service components have to be tested in an efficient and lightweight (i.e., without maintaining dedicated network connections throughout a testing process) manner. Third, how to mitigate the overhead caused by the Web services-specific transport protocols (e.g. SOAP) deserves special attention[1].

## 2. Mobile Service Testing

In [4] indicated that most of the banking apps on a mobile devices act as a front end that invoke services on a back-end server of the bank, which might contact even more servers. Klaus Haller[4] reiterates that the essence of mobile test automation. There are two reasons to automate mobile test cases: to ensure minimal functional coverage and to achieve scalable test configuration coverage. Minimal functional coverage is a safety net. The test scripts cover the basic features of the app and run, for example, each

**Author for correspondence*

night. They cover front-end tests and front-end–back-end-integration tests. A basic test infrastructure consists of a PC running the test script, one local device, and an automation tool. Scalable test configuration coverage checks whether the app runs problem-free on all relevant devices and OS versions. Test scripts must run in parallel on multiple devices, which require a private device cloud with parallelization features.

### 2.1 Mobile Testing Challenges

Following are the mobile testing challenges[5]:

- As mobile apps receive inputs from different providers like users, sensors and connectivity devices, it leads to the unpredictability and high variability of the inputs.
- Languages add some specific constructs for managing mobility, energy consumption and sensing.
- . In order to do mobile apps functional testing, it needs application and the environment like airplane mode, meeting and low battery.
- In GUI testing, the real challenge lies in testing native applications on devices and adequate rendering of data by different devices.
- Mobile device performance and reliability testing depends on the mobile resources, operational mode and connectivity quality.
- Security testing is a major challenge as mobility of the device into networks with different security levels.

### 2.2 The Following are the Criteria for Choosing Mobile Devices

Following are the list of factors to be considered when choosing set of devices for mobile testing[6,7].

- Top supported devices and manufacturers
- Target market (age, geography, business/pleasure/youth, etc…)
- Supporting devices : Smartphones, tablets or both
- Relevant screen sizes
- Relevant regions, carriers and network technologies
- Major supporting OS (i.e. iOS, Windows, Android)[6,7]

### 2.3 Existing Framework

According to [8], Automated testing of distributed, service-oriented applications generally falls into two execution models: centralized and decentralized. In a centralized model, a testing control service sends testing commands to services installed on each host testing, generally in a push model. In a decentralized model, there is no centralized control service, and testing occurs in a purely distributed manner. If synchronization is required, decentralized infrastructures may use network lock files, or they may require separate daemons with intimate knowledge of test sequencing or custom messaging protocols. In this framework, web services are dealt with mobile applications and web application servers. Unlike the above mentioned framework, our framework is dealing with mobile applications with real vehicles communications.

In [9], AppACTS framework was introduced. It is aimed at helping developers to improve mobile application compatibility testing efficiency, save cost and ensure mobile application quality and reliability. Mobile App Automated Compatibility Testing Service (AppACTS) is an online mobile app compatibility testing cloud facility from where end users could upload a mobile app for automated device compatibility testing and view the testing result. AppACTS is based on a scalable architecture that supports to plug in and pull out mobile devices on demand and add or reduced slave nodes and mobile servers at any time.

## 3. Communication between Mobile Devices and Vehicles

The last two decades have seen computer and communication technology become more prevalent in cars, enabling the incorporation of systems and devices for both driver support (e.g., navigation aids) and infotainment (e.g. news and email)[10]. Most of the major car manufactures like GM, Ford, Nissan, Lexus, Audi, etc., use mobile applications as enhanced features in their cars. The mobile applications such as RemoteLink[11], ApplinkSync[12], Audi Connect[13] and myBMW[14] are communicating with owner's cars and control the vehicles from anywhere.

The following are some of the generic operations of the above said mobile applications:

- Makes a secure connection between vehicle and mobile device.
- Start and stop vehicle engines

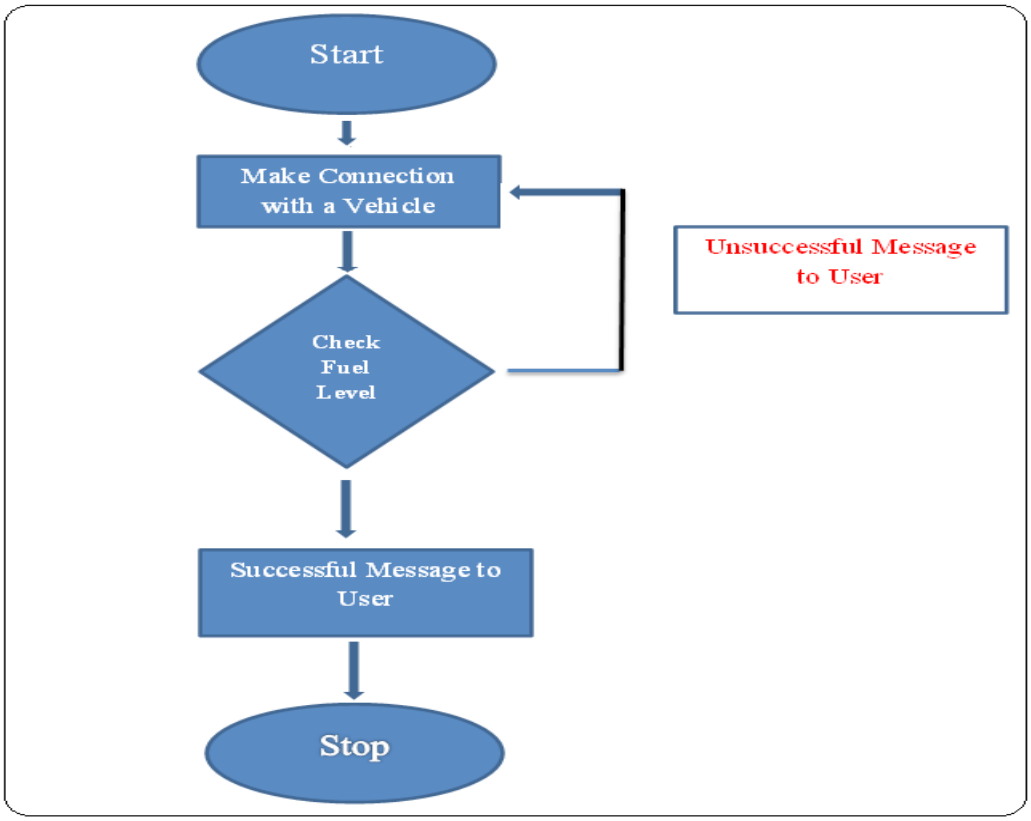


**Figure 1.** MobileVehicle Connection.

- Check real-time fuel and oil levels
- Hands free calling
- Lock and Unlock cars
- Navigation set up
- Vehicle Finder

In order to perform the above mentioned operations through mobile application, we propose a test automation framework for evaluating mobile application's operations whenever they communicate with corresponding vehicles.

The steps involved in mobile vehicle communication are described in the following flow chart in Figure 1.

## 4. Implementation

In order to get some unique identified for the application, the application has to be registered. When user wants to access something on the service, user makes a request to the service using our unique identifier and telling it where to send the user after they authenticate. User will login to the service using the mobile app we launched for that service and choose to grant us application certain privileges. The service will redirect the user to the callback URL that provided and include a code, user can use to get an access token. Then user call the service grants us an access token which user can use to access services user has been granted permissions for. The next time when user needs to make a call, user can just use that access token instead of going through this whole process again.

As shown in the Figure 1, the authorization server validates the request. If the request is valid, the authorization server authenticates the user and obtains the authorization decision from the user. If the user grants the access request, the authorization server issues an authorization code and delivers it to the client.

As shown in the Figure 2, ant build will invoke Java Driver class. As authors K. Vivekanandan, Rajkumar Bhojan, et al, describe in [15], the Java Driver class opens the application, based on the Run Flag status (RUN/NO RUN) given in the data sheet, fetches the data from Data Table and sends it to the AUT (Application Under Test). For better usability, we initiated our service based

automated testing from a functional operation, i.e. functional test script will login to mobile app and clicks on 'Lock' command button on the mobile device. Lock is one of operations to lock the vehicle doors. This request will be sent to vehicle and the response will be received after proper authentication as shown in the screen shots in Figure 3. Once the first cycles gets completed, the controller goes to next record in the Data Table, it searches for Run Flag Status, if it is 'RUN', it executes the second test scripts and so on until it reaches the 'END'. If run flag status is 'NO RUN', controller skips the corresponding test script and moves to the next script.

In this Service based framework, Experitest's SeeTest[16] tool has been used for front end operations of the application. With the help of SeeTest object identifier, we extracted the application elements that we wanted to run a test and stored in object repository. Once it was extracted, the elements were showing up in the Object Repository (OR). Then we incorporate elements to our TestNg framework[17] written in Java. Java scripts were customized for adding features like fetching data from xls sheet and xml files, merging one or more executable methods and sending the reports in html/xslt formats. In this report, we will be able to find out how tests are passed and how tests are failed in a test suite. A sample report table is shown in Table 1. The failed test steps will also have links which redirects to screen shots of the errors captured during the execution. If the request is valid, the authorization server authenticates the user and sends success response to the user. If authentication fails, user will get failed messages from the server.

## 5. Conclusion

In our research, test automation framework has been developed to analyze the web service transactions between mobile application and real vehicles. It has been found that this framework is much simpler and efficient way of

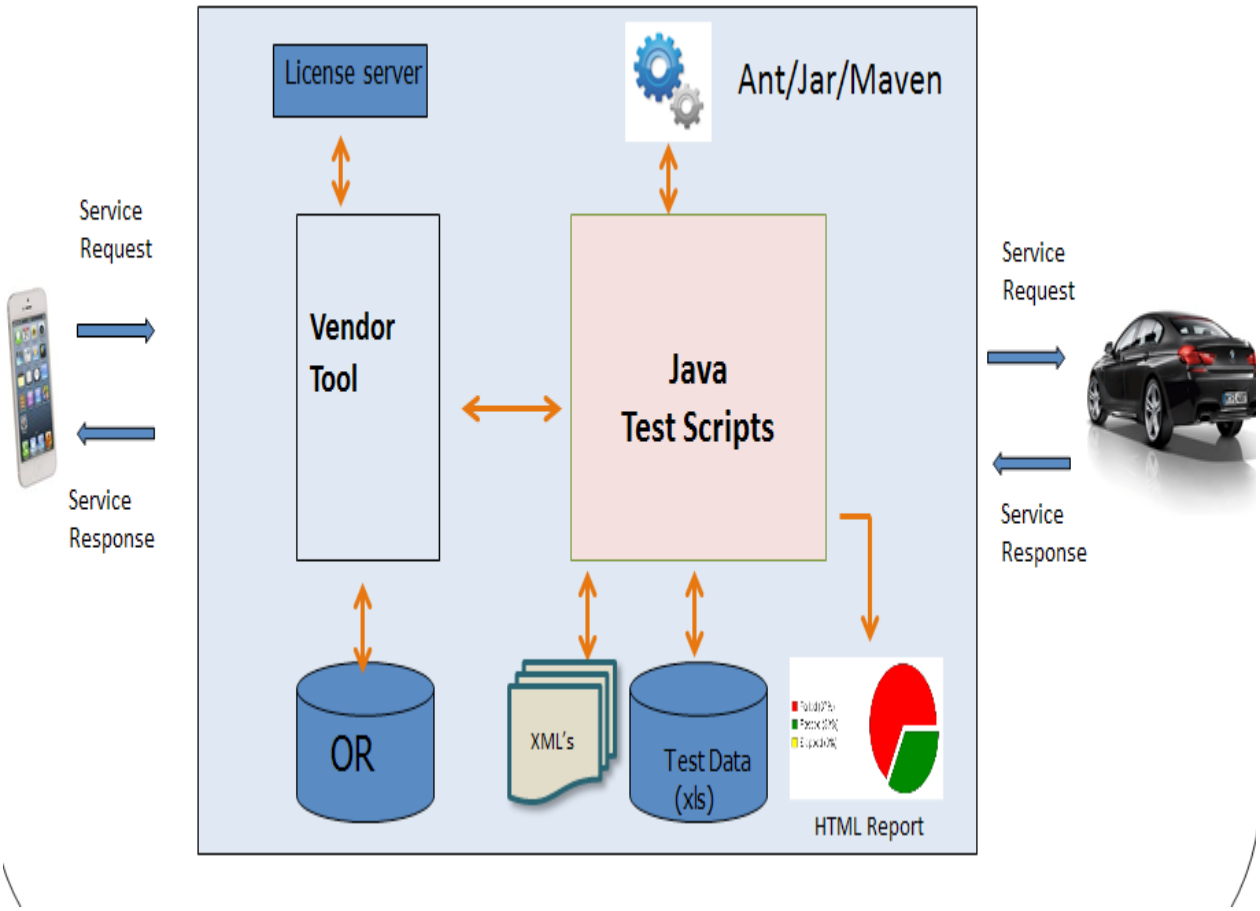


**Figure 2.** Service based test automation framework architecture.

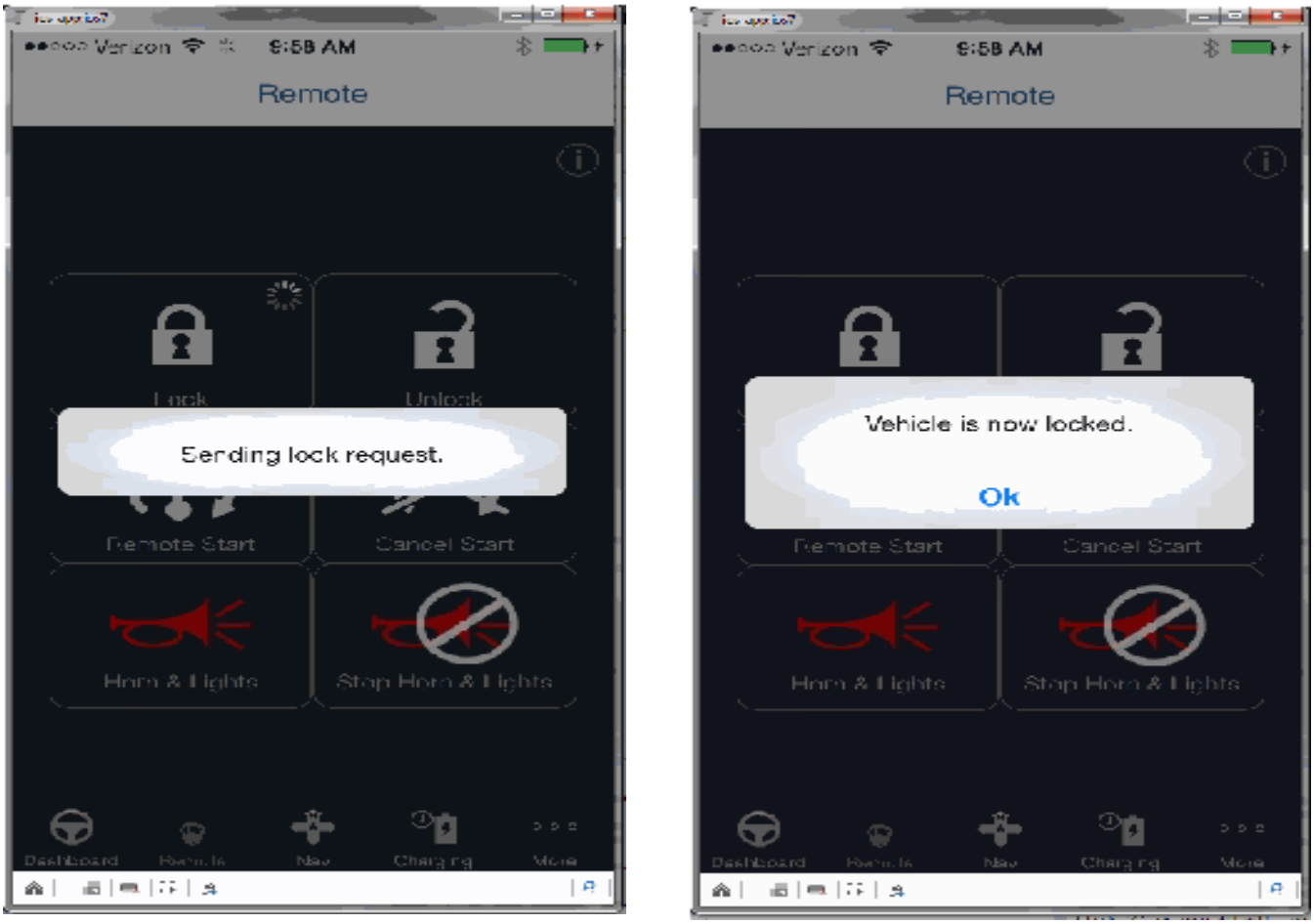


**Figure 3.** Request and response from a vehicle in RLL application.

**Table 1.** Sample HTML Output for Lock Command Response

| Step No | Step Description | Expected Result | Actual Result | Status |
|---|---|---|---|---|
| 1.0 | Login into Remote Link | User must be logged in | User is logged in | Pass |
| 1.1 | Verify if sending lock message appears | Sending lock message must appear | Sending message appears | Pass |
| 1.2 | Verify if 'Vehicle is now locked' alert message is present | Alert must be present | Alert is present | Pass |
| 1.3 | Verify if 'Vehicle is now locked' alert is dismissed on clicking OK | Alert must be dismissed | Alert is dismissed | Pass |
| 1.4 | Verify if lock icon shows correct update date and time | Lock icon must show correct date and time | Lock icon shows correct date and time | Pass |
| 2.0 | Verify if logout is successful | User must return to login page | User return to login page | Pass |

executing regression test suites. Moreover, the existing framework can easily be enhanced for other mobile applications. Our future work includes cloud based mobile testing of the WS transactions standards and their performance evaluation.

## 6. References


1. Zhang J. A Mobile Agent-Based Tool Supporting Web Services Testing. SpringerScience Business Media LLC: 2010 Jan 7. Doi: 10.1007/s11277-009-9879-9.
2. Manova D, Ilieva S, Petrova-Antonova D. Testing Web Service's Compositions Following TASSA Methodology. International Conference on Computer Systems and Technologies (CompSysTech'13). New York, NY, USA: ACM 978-1-4503-2021-4/13/06. p. 185–92.
3. Werner C, BuschmannC, Fischer S. WSDL-driven SOAP compression. International Journal of Web Services Research. 2005; 2(1):14–35.
4. Haller K. Mobile Testing. ACMSIGSOFT Software Engineering. Doi: 10.1145/2532780.2532813.
5. Kirubakaran B, Karthikeyani V. Mobile Application Testing - Challenges and Solution Approach through Automation. International Conference on Pattern Recognition, Informatics and Mobile Engineering. PRIME 2013. 2013 Feb 21-22; Tamil nadu, India.
6. Available from: www.perfectomobile.com
7. Available from: http://www.mbweek.com/2013/07/03/3731
8. Edmondson J, Gokhale A, Neema S. Automating Testing of Service-oriented Mobile Applications with Distributed Knowledge and Reasoning. IEEE International Conference on Service-Oriented Computing and Applications. SOCA; 2011 Dec 12-14; Irvine, California. p. 1–4.
9. Huang J-F. AppACTS: Mobile App Automated Compatibility Testing Service. 2nd IEEE International Conference on Mobile Cloud Computing, Services and Engineering; 2014 Apr 8-11; Oxford; p.85-90. Doi: 10.1109/MobileCloud.2014.13.
10. Irune AA. Evalulating the Visual Demand of In-Vehicle Information System: The Development of a New Method. 2011; 3(1). Doi: 10.4018/978-1-4666-2068-1.ch001.
11. Available from: www.onstar.com
12. Available from: http://support.ford.com/sync-technology/applink-overview-sync
13. Available from: http://www.audiusa.com/innovation/intelligence/audi-connect
14. Available from: http://www.bmw.com/com/en/owners/bmw_apps_2013/apps/my_bmw_remote_app/index.html
15. Vivekanandan K, Bhojan R, Ganesan S. Cloud Enabled Test Evaluation on Mobile Web Applications. International Journal of Advanced Research in Computer and Communication Engineering. 2014 Jun; 3(6).
16. Available from: http://www.experitest.com
17. Available from: http://testng.org/doc/index.html