SIP Automated Test Platform

Yagmur Kirkagac
Netas Telecommunication Corp.
Marmara University
Electrical and Electronics Engineering
Istanbul, Turkey
Email: yagmur@netas.com.tr

Serdar Simsek
Kocaeli University
Computer Engineering
Human Computer Interaction Lab.
Kocaeli, Turkey
Email: ssimsek@outlook.com

Demir Y. Yavas
Netas Telecommunication Corp.
Istanbul Technical University
Electrical and Electronics Engineering
Istanbul, Turkey
Email: demiry@netas.com.tr

***Abstract*—IP networks have been becoming more popular communication infrastructures due to the lower operational costs in recent years. Mainly, usage of IP networks in voice services, which is denominated by VoIP (Voice over IP), has made difference in telephony networks not only the way of the people's communications, but also telecommunication companies' and operators' solutions. In this scope, IMS (IP Multimedia Subsystems)/SIP (Session Initiation Protocol) networks have some challenges for testing processes. Testing and validation of SIP scenarios in IMS networks can be complex and time consuming due to the lots of different kind of scenarios with traditional manual methods. This paper aims to develop automated test environment for SIP messages. In this perspective, this system provides to manage SIPp-like XML (eXtensible Markup Language) scenario files, which is aimed to support IMS network nodes with GUI (Graphical User Interface). Automation of SIP signaling test environment is also implemented for plotting data automatically.**

***Keywords*—SIP; SIPp; Automated Test Environment; GUI; VoIP; IMS**

I. Introduction

IMS provides the voice transmission over IP to merge all networks for IP based communication. VoIP solutions also use SIP for session management [1], and RTP (Real-Time Transport Protocol) for multimedia transmissions [2]. SIP is one of the commonly used protocol for session management of video and voice transmissions in recent years.

Verification and validation tests are important part of the product development life cycle and can give an idea about the quality of the product. Testing can either be done manually or using an automated testing tool. Manual testing is extremely time-consuming and error-prone process. It is common expectation that automated testing saves resources and makes testing process more expedite, efficient and reliable. In addition, automated testing can be a part of CI (Continuous Integration). More generally, the correct and efficient mapping between requirements and test cases and full implementation of them might be time consuming and difficult to reach [3].

More specifically, testing SIP protocol has some challenges valid for both manual and automated testing approaches. Although the text based message format simplifies interpretation messages and message flows; when it is combined with rich feature set of SIP, dynamic behavior of certain message headers (such as ”Via” and ”Contact” headers), the large diversity of header field values, the testing becomes a complex and error-prone task. In detail, the header parameter interpretation may differ according to their precise location within a specific message flow [3].

In the case of sequentially invoked test cases for a specific user or service, a failure on a test case may lead the SUT (System Under Test) to inconsistent states that subsequent tests cases may also fail. This may cause difficulties on determining actual problems. In this context, the automated testing tools are needed to recover failure cases (such that sending CANCEL or BYE SIP messages in order to end the current session as depending on the phase of the session).

Support for RTP with the test tool is a valuable facility to be able to verify media path establishment and call scenarios including SIP-based media servers.

If a test tool supports protocols, such as HTTP (Hyper-Text Transfer Protocol), SOAP (Simple Object Access Protocol), to be used change the configuration of the SUT dynamically, it can be more effectively used in a automated test process.

On manual testing and during the test case development, CLI (Command Line Interface) might be challenging for testing process even if it has some advantages. CLI allows to experienced testers to get in contact with the system as quickly as possible. On the other hand, CLI has lots of problems for testers. Supporting GUI does not make only creating test cases easier but also providing accelerated and efficient testing time.

All these kind of issues with combining some others such as incorporating with the new additions to specifications and maintaining also the backward compatibility make providing a comprehensive test tool a challenging task.

In this paper, we presented a test tool and a test platform addressing these kind of challenges. The test tool supports SIPp-like XML-based scenarios, test cases covering test scenarios and test suites covering test cases. Supporting SIPp-like scenarios is selected to capitalize on habits of testers who have already known about SIPp [4]. Besides, properties provided by the SIPp approach are extended to provide complex call scenarios including multi-dialogs and nested transactions. The test platform provides sharing applied test structures through a database and reorganization between teams taking role at

different testing phases such as compatibility testing, sanity testing, regression testing and acceptance testing, except performance tests which is not assumed for the tool. In this way, we plan to build a corpus of SIP messages containing a rich mix of requests and responses across different services. The presented tool supports sending media stream by using media files, and it can generate and recognize both inband and outband (RFC 2388) DTMF (Dual Tone Multi Frequency) tones through RTP streams [5].

The tool can simulate SIP client and SIP server behaviors including, but not limited to proxy server, B2BUA (Back-to-Back User Agent), registrar, SIP media server.

The proposed tool offers the utilization of powerful SIP test scenarios by using the proposed methodology which is explained in Section II. The technologies, used in our tool, are explained in Section III and supported graphical user interface is explained in Section IV.

II. Methodology

A. Testing Objectives

As testing is an integrated part of a product development, it is necessary to perform testing at any phase of the development life cycle for distinct objectives. Our tool and platform is aimed for *(i)* design phase tests (e.g. as an message injector), *(ii)* compatibility testing, *(iii)* sanity testing, *(iv)* regression testing, and *(v)* acceptance testing. It is planned to be used for reproducing of reported problems and collecting required debug information at customer sites. The tool is not assumed for performance tests. The tool can be used for manual testing as well as integrated with CI for automated real time tests.

B. High Level Design

The high level design of the tool is done by considering the following criteria:

Simplicity: The tool supports minimal menu options and configuration details for ease of use in user interface (UI) design. All test components are movable with drag and drop design mechanism. The tool is natural design considering human-machine interactions and user profiles. Testers can learn without experience the tool easily, and use the system quickly. New testers become productive in a short time.

Reusability: Our designed tool supports to reuse, change and improve all test components such as test messages, test scenarios, test cases to another test.

Portability: Our designed tool allows to run the same tests with another platforms which are another SIP enabled servers or clients with simple configuration changes like IP/userID.

Flexibility: All test components can adapt all processes without cause of any flow damages.

Collaboration: Our tool design encourages the sharing and collaboration. It allows to share test components with other testers easily. Testers can search test components not only in our created library but also other testers working areas.

The high level design components are listed below.

C. Design Components

1) User Login and Credentials: There are user login window for each tester to sign in. After the tester signing in, they do not have to memorize the script codes for functionalities which are founding in RFC. Also they do not have to re-write the codes over and over again. Testers can try the test scenarios with the constituted default test cases. Moreover, testers can add and/or save their specialized test scenarios into the system. In fact, they can edit or pull the pieces the scenarios. With this approach the testers can share their experiences about testing cases and improve their methods. Every user has a different account in order to accomplish those facts laying in previous sentences. Test message templates can be edited only by its creator. Otherwise, a clone is transferred to the user who is attempting to save. After the clone, the message will be appeared other users pool.

2) Standards/Specifications: The testing and developing process in VoIP technologies, is necessary to provide IETF (Internet Engineering Task Force) and 3GPP standards. Designed tool is supported by multiple protocols like SIP, RTP, SOAP, HTTP [6] and also is used in network transport protocol layers TCP, TLS/TCP or UDP [7].

3) Platform Independence: For platform independence, the tool is implemented by using Java and tested on Windows and Linux OSs.

4) Extendability for Multi-protocol Support: An event driven core mechanism is implemented as independent of protocol messages and flows. The event driven mechanism follows only events defined in the scenario flow such as message sending, message receiving and time expiry events. A new protocol definition can bind itself to the core mechanism to invoke and get events.

5) Multi-agent Support: As multi-agent support, the tool can form itself as a client or a server as depending on the first action in the scenario; e.g. if the first action is to send a request message, it considers a client behavior.

6) Mode of Operation: The tool considers three mode of operation on message processing. It parses messages according to SIP ABNF (Augmented Backus-Naur Form) grammar before sending through SIP transaction and dialog state machine by default. This is to prevent uncontrolled message structure and flow errors. The advanced mode controls only mistakes only message grammar before sending, but not transaction and dialog states. JAIN-SIP stack [8] is used for these purposes. The advanced mode uses only the message parser of JAIN-SIP stack. The more advanced mode does not perform any control on message structure and flow and it uses an internally developed stack. This is mode can be used to produce malformed messages for stress and robustness tests.

7) Test Scenarios Setup: All test scenarios are grouped under a specific functionality by specific users. Test scenarios lists can be grown by adding more scenarios from testers. Our designed tool does not need any configuration to setup for

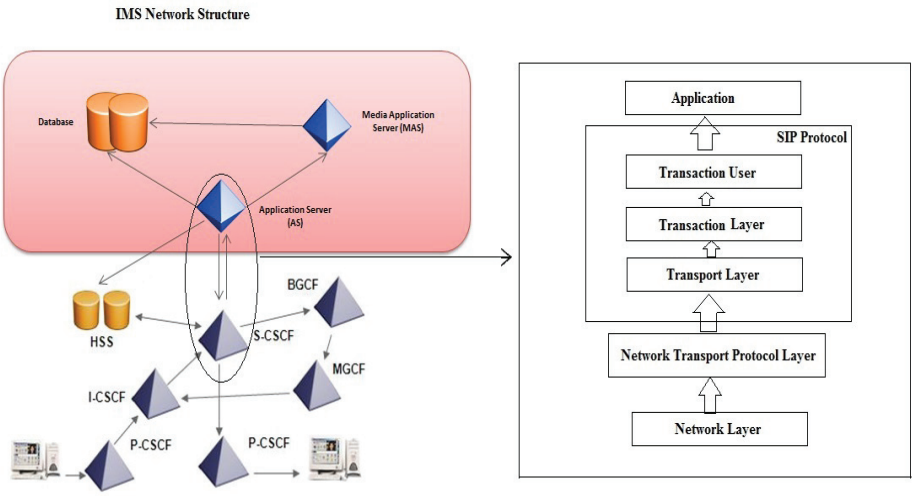

Fig. 1. SIP Protocol Stack in a Typical IMS Structure

test scenarios. User can use test scenarios only drag and drop method.

8) External Tools: Our designed tool does not need any additional tools for examining tests. It provides a log screen at the bottom of the window that testers can see messages and exceptions in this part of the tool. Packet analyzer tools like Wireshark still can be used for network troubleshooting.

9) Test Scripts: Basic test scripts are developed by in-house testers which are denominated by admin users in designed test tool. Other additional test scripts can be specified by other testers. Stress testing is more difficult without automated testing tool. Designed tool supports 5 testing types which are compatibility, sanity, regression and acceptance including stress testing automation [3].

10) Verification: Creating reliable and functional SIP signaling testing tool is quite challenging. There are lots of parameters which are depending on protocol failing (when deployed) even after testing process. Tool design should also provide verification with automatically. To explain the real-case example, call establishment stage can be failed in VoIP testing. Other testing tools do not give any information to indicate error. Testers spend a lot time for reproducing problem and identifying error. Designed tool detects any problem when the call signal is failed in the call process and inform the tester which stage is failed. By this means, testers can identify problems easily and fixing time also will decrease.

11) Test Case and Test Suite: Test cases consist of scenarios which defined a complete test case, such as grouping scenarios of originating and terminating agents in a test case which can be invoked at the same time from the same computer. For flexibility, scenarios in test case can be invoked at the same time or sequentially. Similarly, a group of test cases can be grouped into a *test suite* which can be correspond to a specific feature or functionality test. Our test suite is designed to run multiple test cases ordered one by one execution, or a group of test cases simultaneous.

12) User Friendliness: A well defined GUI (Graphical User Interface) is considered for user friendliness. The user can build a test scenario by drag and drop from the entire corpus provided by the test platform using a database. The GUI provides facilities for users to edit, re-configure, run tests and to collect results.

13) Graphical Display: The tool has a graphical interface providing operational management (OM) statistics for every monitored network element (NE). The graphical display is supported by OM data with a specific time interval. In addition, it plotting graphs with supports available data sets. Thus, testers not only can detect problems of OM data but also manage fault performance with using graphical display easily.

14) User Interface: The GUI software testing is promised to the graphical front-end meets with tool specifications. GUI software testing must provide not only cover all functionalities but also ensures that the GUI itself is fully tested.

III. Test Platform Design Technologies

IMS network structure and SIP Stack structure considered for the tool is illustrated in Figure 1. The tool can be positioned as any SIP enabled node in the network such as SCSF, AS, SBC, SIP GW, SIP Clients [9]. JAIN-SIP and an internally developed stack are used in the SIP stack level. The internally developed stack is also used for RTP, HTTP and SOAP.

Designed tool is a desktop application for testers. High level design is required user-friendly design which is mentioned in

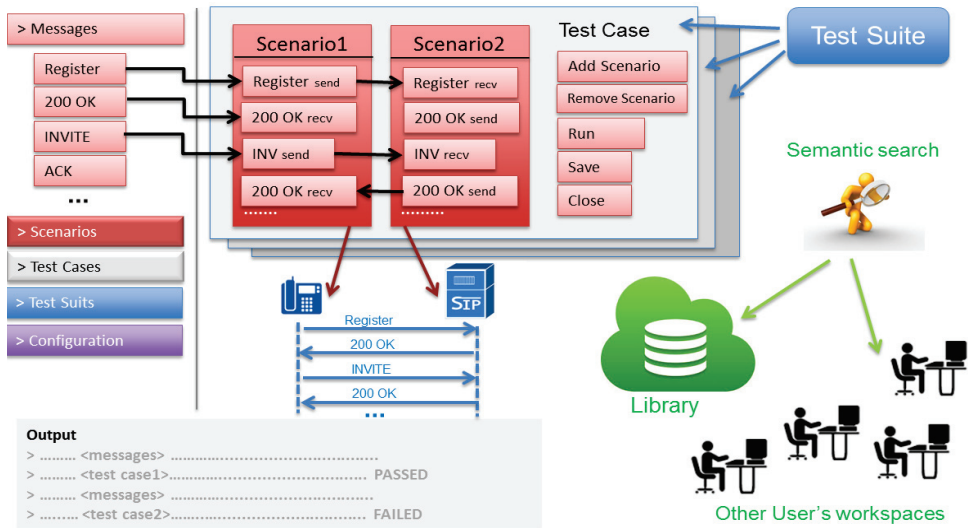

Fig. 2. Designed Tool Overall System Design

section II-C. It has been developed using JavaFX for building user-friendly GUI. It is developed with using Model-View-Controller(MVC) design pattern.

MVC design pattern has three types of objects. There are; model, view and controller. These objects can handle entity (data), boundary (presentation) and control (behavior). View and controller objects compose user interface. MVC model has useful advantages at runtime such as providing multiple views, synchronization of views and controllers. Designed tool is used to model object to operate and implement the logic. View model object is used to interact with testers. Control object is used to control the web applications.

The back-end software of the tool is implemented in Core Java. Database system is developed in LAMP (Linux + Apache + MySQL + PHP). LAMP server use script languages such as PHP, Perl, JSP or ASP. These are quite efficient for quick learning and deployment. Core Java objects are connected with relational database MySQL by using Java Database Connection. In this means, all parameters are placed in relational database MySQL.

Mapping is one of the most important part in designed tool. Mapping allows to the testers to map the test cases with users. In other work, our designed tool allows to semantic search in library or other users workspaces for test cases. Designed tool is supported Object Relational Mapping (ORM) design which is provided a framework for mapping an object oriented model design to relational database. ORM design is mostly used with Hibernate. We designed unique ORM framework design for mapping objects to relational database. Original ORM framework design was carried out in this tool.

Designed tool overall system schema is illustrated in Figure 2. Our tool supports multi-protocol (SIP-RTP-HTTP-SOAP) in the same scenario. Testers can search libraries and other users workspaces with semantic search method.

Race condition problems, which may not be reproducible in real time, can be created by the with extending test scenarios by adding timers.

IV. Graphical User Interface

The GUI has been developed in JavaFX for improving the user-interface experiences. User account is an important parameter which is provided sharing, utilization of scripts in designed tool, in that scope testers can learn and drive testing processes much more easier way. In this section, designed tool sample screens will be shown and will be given brief information.

The tool provides a user login screen. Testers create an account first. The tool allows to semantic search for test scenarios searching in our created library or other testers workspaces. Testers can define their SIP messages with filling special fields. Admin users can also share messages in the library as a template. Other testers can also add created test objects into the library as a template, after verifications. If testers do not share their test objects to the library, test objects can be visible on only themselves workspaces. However, our semantic search also allows to find test objects even testers do not share to the library. All test objects can use with cloning testers workspaces.

Template messages can be added another users pool. The user can change the messages and save specialized message to own pool without any changes in original template messages. We also implemented chat screen for providing a communication of testers about testing processes and templates. Figure 3

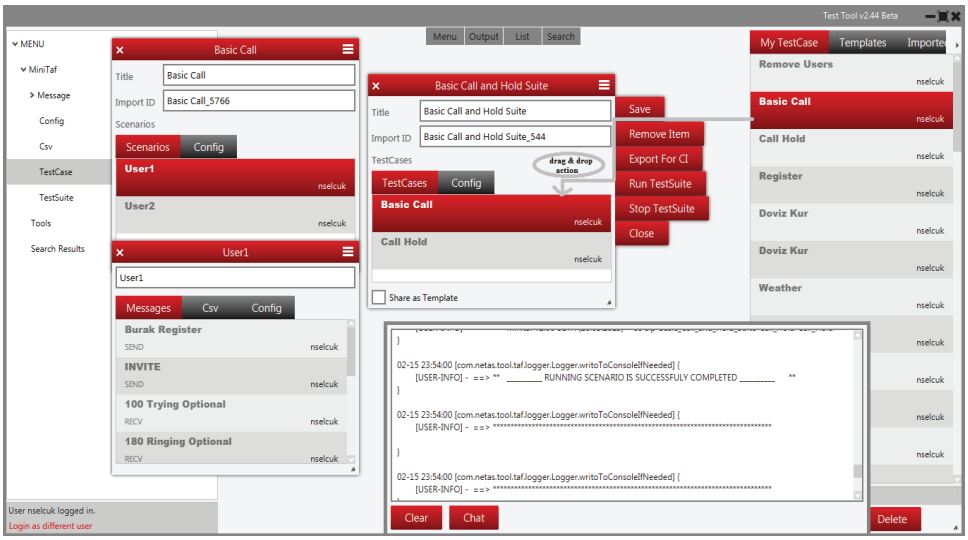

Fig. 3. Designed Test Environment Test Suite Example

shows our designed test environment which is supported with basic call and call hold test suite example.

The tool plotted graphics depends on customer data automatically for catalyzing the analysis. It also allows to add another features in graphics for making easier to comparisons and analyses. System can analyze two or more parameters in the same graph for making easier to comparison.

V. Conclusion

In this paper, we present our automated SIP testing tool and platform with solutions for SIP testing problems. These problems are solved with supporting multi-protocol structure, multi-platform computing methods, complex SIP call scenarios, user friendly and collaborative GUI design. All of our solutions consider not to change users's habits but simplify. For this reason, our tool design supports XML syntax which has already known by testers. Our designed Test Case and Test Suite structure can handle complex SIP call scenarios by running the organized more than one test scenarios sequentially. The tool supports protocols such as HTTP and SOAP for changing the configuration of the SUT during the tests, dynamically. The tool consists of an user-friendly GUI design. Drag and drop feature simplifies the usage and learning the tool. It plots graphs from data and runs the stored scenarios from the library which is shared other tester's scenarios pool. These are the advantages for the cases which have to be analyzed and tested immediately.

In the future works, our tool can be expanded for involving all IMS network protocols by adding features such as supporting Diameter protocol and RESTFUL services. In addition, we desire our tool behave as a traffic tool with supporting complex call scenarios.

Acknowledgment

This project has been supported by Netas Telecommunication Corporation.

References

[1] J. Rosenberg et al., SIP: Session Initiation Protocol, RFC 3261, June 2002.
[2] H. Schulzrinne, et al., RTP: A Transport Protocol for Real-Time Applications, RFC 3550, July 2003.
[3] D. Goncalves, A. Amaral, A. Costa, P. Sousa, "Towards Automated Test and Validation of SIP Solutions", Telecommunication Systems March 2016, Volume 61, Issue 3, pp. 579-590.
[4] SIPp development team, "SIPp - SIP performance Testing Tool" [online] Available: http://sipp.sourceforge.net/.
[5] L. Masinter, Returning Values from Forms: multipart/form-data, RFC 2388, August 1998.
[6] P. Subramanian, B. PG, "Convergence of Java EE and SIP in IMS AS", IP Multimedia Subsystem Architecture and Applications, Int. Conf., Bangalore, 2007, pp. 1-5.
[7] M. Ranganathan, O. Deruelle, D. Montgomery, "Testing SIP Call Flows Using XML Protocol Templates", TestCom'03 Proc. of the 15th IFIP Int. Conf. on Testing of Communicating systems, pp. 33-48
[8] The Source for Java Technology Collaboration, "JAIN-SIP Stack" [online] Available: https://jsip.java.net/.
[9] M. Poikselka and G. Mayer, "The IMS: IP Multimedia Concepts and Services", 3rd Ed., Wiley, 2009.