Automated Testing of a Converged Conferencing Application

Venkita Subramonian, Eric Cheung, Gerald Karam
AT&T Labs Research,
Florham Park, NJ, USA

Abstract

In this paper, we describe our experience with automated testing of a mission-critical internal Voice-over-IP (VoIP) conferencing application which presents a web interface as well as a voice interface. We document the challenges that we had to overcome when testing this application and then present our solution using open source testing tools. The lessons learned from this experience may be applicable to a broad class of applications that pose similar testing challenges.

1. Introduction

Automated testing for web based applications has grown increasingly complex and challenging due to a wide variety of technology available to create and deploy web applications. The convergence of web and IP-based multimedia services poses a new set of challenges for testing *converged* applications. These applications typically deal with multiple protocols — *e.g.,* Session Initiation Protocol (SIP) [5], HTTP [4], RTP [6], email. Apart from multiple protocols, converged applications also deal with multiple user interfaces — *e.g.,* web, telephony, chat client. For example, a subscriber can access her ISP web portal and initiate a voice/video call to another party. In this case, an HTTP request over the web invokes an IP-based multimedia application, which then initiates a voice/video session between the subscriber and the called party, possibly through SIP and RTP. Similarly, a multimedia session could initiate a web service that ultimately updates a web browser with status related to the multimedia session. This case study discusses the challenges we faced while automating functional testing for a converged IP-based conferencing application and how we overcame those challenges.

2. The System Under Test

The System Under Test (SUT) is a large-scale VoIP conferencing system that provides the primary conferencing service within the company. Deployed in 2007, it currently supports on the order of 10,000 simultaneous calls, and handles millions of call minutes on a typical workday. Besides basic conferencing, the service offers a number of advanced features, in particular seamless combination of the telephony and web experience. A typical user experience is as follows:

- A user logs into the service web site and navigates to a current meeting. The list of participants who are already on the meeting, as well as those invited but not yet joined, are shown. The user can either call in or ask the system to dial a preset or new number.
- When this user joins, all participants hear a join tone, and their web pages are updated to show the new arrival.
- During a meeting, a participant can use the phone keypad or the web page to invoke features, for example to mute or unmute the line. In either case, every participant's web page is updated to show the new status.
- The meeting host has additional features, e.g. to mute or drop any participants, or to terminate the meeting.

Figure 1 shows a simplified architecture of the conferencing system. The web application executes on standard Java Servlet and JavaServer Pages application servers, and the SIP application executes on standard SIP Servlet application servers. Media servers provide the audio mixing as well as Interactive Voice Response (IVR) functionality, and Public Switched Telephone Network (PSTN) to SIP gateways convert the PSTN calls to VoIP. There are multiple instances of each of these components to provide load sharing and

AST'09, May 18–19, 2009, Vancouver, Canada
978-1-4244-3711-5/09/$25.00 © 2009 IEEE

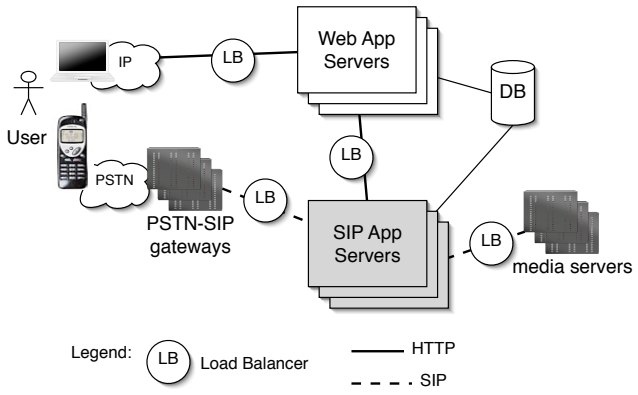

Figure 1. Architecture of the SUT

high availability, and HTTP and SIP load balancers are used to distribute the traffic among the instances.

As with all complex systems, the conferencing system is susceptible to failures ranging from software defects and component overloads to hardware failures. Although the system is designed with certain self-diagnosis and alarm functionality, as well as to degrade gracefully when failures occur, there are occasions when users are affected without prior warning to the operators. There are also situations when it is hard to detect which component and which instance has a fault, especially as the load balancers can mask a faulty instance.

When the system was first deployed, the operators relied on a set of manual test procedures to ascertain the integrity of the deployed system when: (1) an alarm was raised and after it cleared, and (2) problems were reported by users. This manual testing was time and resource consuming, and delayed failure detection and corrective actions. It became clear that an automated test tool could provide frequent, proactive testing to detect problems more quickly, and hence reduce the duration of degraded operations. Automated post-deployment testing could also test each component individually and isolate faults more rapidly.

3. Testing Challenges

The following is a simple test scenario that exercises the main components in the conferencing application and performs a functionality test. This closely reflects the typical manual test procedures that were used for post-deployment testing of this application. It constitutes a "sanity" test since it provides black box testing of representative, but most essential behavior of the application, as experienced by an end user.

```
Login into web portal using corporate single sign-on
Host creates a conference bridge through her web portal
Grab the new conference url
Participant logs in to her web portal
   and navigates to the above conference url
Grab host and participant PINs from conference page
Host dials in to conference bridge
Host interacts with IVR using touchtones
   - enter userid followed by PIN
Participant dials in to conference bridge
Participant interacts with IVR using touchtones
   - userid followed by PIN
Verify conference web page shows both joined to conference
Host speaks. Verify participant can hear audio.
Host ends meeting through web
Verify host and participant audio sessions ended
```

To perform this test manually, a tester has to play the role of the host and the participant and thus has to deal with the challenging task of handling four endpoints – two browsers and two VoIP endpoints. This simple example motivates the need for a test tool that allows us to automate functional testing of converged applications. Automation of this test raises the following important challenges for a test tool:

Emulate multiple protocol endpoints To realize a converged test scenario such as the one described here, a converged test tool is expected to have the following capabilities:

- Web testing: Web browser emulation; traverse an HTML document and search for content; support for client-side Java technologies such as AJAX and Javascript that are used in the application
- SIP testing: SIP/RTP endpoint emulation; support for IVR interaction such as sending keypad events; sending and receiving media

Reconcile message exchange characteristics of multiple protocol endpoints A converged test tool should be able to deal with multiple protocol characteristics. In HTTP, an endpoint is exclusively either a client or a server – the client initiating an HTTP request and the server serving a response back to the client. In contrast, SIP/RTP is a peer-to-peer protocol where an endpoint could be both the originator and receiver of requests and responses. A single request from a SIP endpoint could result in multiple responses.

Coordinate execution of test endpoints Another big challenge is to coordinate the execution of protocol endpoints and enable easy information exchange among them. For example, after creating a conference through the web portal, a test tool has to extract the host and participant PINs generated and displayed by the application. These PINs are to be used later by the

SIP endpoints for dialing into the conference bridge. A requirement such as this creates dependencies among the execution of endpoints and hence requires strict synchronization of test execution across different protocols. Such synchronization is hard when a single test scenario specification is realized using multiple concurrently executing test endpoints as is the case with many existing test tools.

4. Our Solution

We initially surveyed existing tools for SIP and web testing and evaluated their capabilities against the challenges described in Section 3.

Although existing SIP test tools [7] offer a variety of features and capabilities for low-level protocol testing, we found them cumbersome to use for functional testing of converged VoIP applications. In fact, we had originally partly automated the SIP part of the test scenario described earlier using the popular SIP test tool SIPp [8]. Test scripts in SIPp are XML based and consist of SIP messages to be sent or received by test agents. Multiple such test scripts are written and executed to simulate multiple test agents. Moreover, to simulate keypad based IVR interaction (*e.g.,* entering conference userid and PIN) in SIPp, one has to first perform this interaction manually and record the RTP audio session (using packet capture tools such as wireshark) between a VoIP agent and the SUT. This packet capture file contains the key entries by the user and is then later used as part of a SIPp script to replay the keypad entries. In the conferencing example scenario, it is difficult to capture and replay the conference PINs for IVR interaction since the PINs are generated dynamically.

We also explored usage of TTCN-3 [11] for our test automation. This is a test specification standard that allows creation of abstract test cases using the TTCN-3 language. While there are commercial vendor offerings for SIP-specific TTCN-3 codecs and adapters, there is little information available about their applicability to testing converged applications. Moreover, open source implementations of TTCN-3 runtime, codecs and adapters to support converged application testing have been hard to find.

We found SipUnit [9] to be the closest in terms of being able to host multiple endpoints within the same test case. However, SipUnit does not have support for RTP, which made it an unsuitable candidate to automatically interact with an IVR.

To satisfy our test automation requirements, we developed the KitCAT (Kit for Converged Application Testing) framework [10], which is a Java library for testing SIP/RTP applications and is released open source. A KitCAT test agent is used to simulate a SIP/RTP end user of the SUT and thus responsible for sending and receiving SIP/RTP messages. An agent also maintains state which is advanced appropriately as SIP/RTP messages are sent and received by that agent. During the course of a test, a test case can control execution of agents and perform condition checks on the state as well as messages sent/received by agents. An agent can act as either a caller initiating SIP/RTP messages to the SUT or a callee receiving SIP/RTP messages from the SUT. A KitCAT test case hosts both caller and callee agents within a single process, thus allowing easier coordination among the caller and callee agents.

For web testing, we found HtmlUnit [2] which is an open source Java library that provides an API for browser emulation, page navigation, and content search in retrieved HTML pages. It also includes support for Javascript and AJAX.

Figure 2 shows our solution test architecture for converged applications. The test endpoints are hosted in a JUnit [3] test case, which is the de-facto standard framework for writing and running automated tests in Java. Using JUnit enables us to leverage existing support for JUnit based testing. For example, this enables converged application developers to easily run these test cases right from their development environment that has support for executing JUnit test cases. Moreover, software build tools such as Ant [1] have built-in support for running JUnit-based test suites and generating formatted test reports.

The combination of KitCAT and HtmlUnit provided us with a powerful solution that adequately addressed the challenges described in Section 3. Our solution enables emulation of multiple SIP/RTP and browser endpoints within a single Java test script. KitCAT agents are designed in such a way that their execution is controlled by agent-specific test case instructions as well as SIP/RTP messages arriving from the SUT. KitCAT also provides primitives to control the concurrency of execution of agents. In the concurrent execution mode, the test case execution itself proceeds concurrently with agent processing of SIP messages. This execution mode is undesirable when the test case needs fine-grained control over agent execution. Such fine-grained control is supported in the non-concurrent execution mode where SIP messages are processed by agents only at designated points as specified by the test case. The test case thus acts as a controller orchestrating the activities of different protocol endpoints. Since all endpoints are hosted inside a single test case, the test case can fetch data obtained from one endpoint

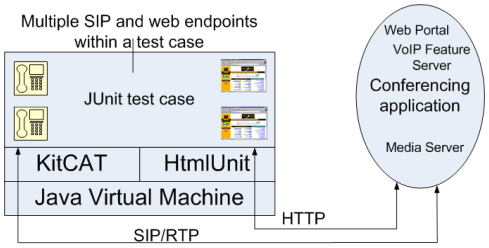

Figure 2. Automated Test Architecture

```
SIPAgent host = createAgent("Host");
hostBrowser = new TMeetBrowser();
hostHomePage = hostBrowser.login(hostId, hostPasswd);
hostMeetPage = hostHomePage.createAdhocMeeting();

hostUserName = hostMeetPage.getLoginUserName();
bridgeNum = hostMeetPage.getBridgeNumber();
hostUid = hostMeetPage.getPhoneUID();
hostPin = hostMeetPage.getHostPin();
partPin = hostMeetPage.getParticipantPin();

host.call("sip:" + bridgeNum + "@" + sipServer);
processSIP(10000);
assertThat(host, is (connected()));
host.sendDTMF(hostUid + "#");
processSIP(3000);
host.sendDTMF(hostPin + "#");
processSIP(20000);
assertThat(host, is (connected()));
processSIP(30000);
assertThat(hostMeetPage,
                   participantJoined(hostUserName));
hostMeetPage.endMeeting();
processSIP(30000);
assertThat(hostMeetPage, has (ended()));
assertThat(host, is (disconnected()));
```

Figure 3. Excerpt from an actual test case

and pass it along to another endpoint - *e.g.,* getting the conference PIN from the browser endpoint and using that in the SIP endpoint.

For the conferencing application test automation, we also constructed application-specific wrapper libraries based on HtmlUnit. The library contains primitives that include web portal navigation (*e.g.,* create adhoc meeting, end meeting, upload shared document, mute participant, remove participant), content retrieval from returned HTML page (*e.g.,* get conference PINs for host and participant) and condition checks (*e.g.,* is participant shown as joined to a conference). An excerpt from the resulting test script showing only the host related activities is shown in Figure 3. The actual test script contains similar participant activities interleaved. The resulting test script closely mimics the example script shown in section 3.

Effectiveness of our solution: We created an automated test using KitCAT and HtmlUnit to perform frequent functional tests for our conferencing application. To perform proactive automated tests as mentioned in section 2, we run periodic (every 5 minutes 24×7) functional tests directly against each SIP application server. Test calls are also placed to the PSTN gateway, which simulate users dialing in. The results of these tests are collected and published on an internal website. Apart from PASS/FAIL results for each test, this site includes SIP message logs as well as audio that was received by the SIP/RTP endpoints during the test. These logs are helpful for diagnostic purposes in case of test failure. This web site is constantly monitored by our production support staff. Alarms are triggered based on reported failures so that preemptive action can be taken. The periodic automated testing has thus become a critical part of production monitoring of the conferencing software.

5. Lessons Learned

Combining several tools was necessary to provide an effective solution Our experience with testing converged applications has shown that a combination of tools are necessary to meet their testing requirements. The combination of KitCAT and HtmlUnit within a JUnit test case has provided us a powerful solution, both in terms of increased effectiveness and, due to the use of open source methods as well as ease of authoring, the lowered cost. So far, there have been over 1000 known test cases that have used our solution. These test cases are used in regression-testing applications as part of multiple internal projects. We intend to conduct further user studies to evaluate the impact of our solution on reduction of testing time in an entire project life-cycle.

Post-deployment tests have to be resilient to test errors For post-deployment tests, the test setup may sometimes get corrupted due to a test error. The test script should be resilient to such errors and restore the test setup. For example, in the post-deployment tests for the conferencing application, a test error may cause a test conference to be corrupted as a result of which subsequent tests produce false failures because of the inability of a host or participant to dial in to a conference bridge. Automatic recovery strategies had to be put in place to detect and recover from such situations.

Application-specific web testing primitives aid re-usability and readability Even though HtmlUnit provides rich content search capabilities within an HTML document, it is useful to have application-specific primitives that hides details about how a par-

ticular functional testing task is achieved — *e.g.,* click on a particular meeting, upload a shared document, mute a particular participant, check if a participant is shown as joined to a conference. The advantages of developing these primitives are: (a) while these primitives may not be reusable across applications, they can be reusable across different test cases for an application; and (b) better test case readability. Based on the current experience, we believe that the utility of higher-level wrapper primitives is best decided on an application-by-application basis.

Enhancing presentation content in web applications with testing-oriented hints aids in functional testing of applications Relying on web page content for our testing has two inherent challenges — (a) tests may rely on implementation details such as the structure of the HTML document in a web page and (b) they may be hard to write manually since it would require understanding the structure of the content within application web pages. For example, in our conferencing application tests, we use XPath [12] search expressions to determine the existence of a particular HTML image element in the document hierarchy of a meeting web page indicating that a participant has joined a conference. Such dependency may make these tests fragile in the face of implementation changes in the structure or content of the concerned HTML documents. To reduce the impact of implementation changes, it would be helpful to incorporate "hints" as part of web page content that would aid testing. For example, the meeting web page could embed a hint in the form of an invisible HTML table that contains the state information of each participant. Both the web page presentation logic and the tests would then depend on these hints. Further any changes to the hints would immediately, and possibly automatically, identify possible changes needed in the tests. This will work only if one has control over the source code for the application. We plan to explore this topic further as part of future research.

Limitations of our approach Although our solution has been effective for converged application testing as part of internal projects, it has limitations some of which are listed here. (a) It does not provide a general solution to compose arbitrary test tools to form a converged testing tool. KitCAT and HtmlUnit are both Java-based and that commonality helped us avoid challenges such as reconciling differences in test scripting language and output format for test results. (b) Our solution is currently restricted to a Java-based runtime environment. (c) Our solution would need further work with respect to test case generation. Testers, as opposed to developers, may not have application programming skills; they define and execute test cases, using tools developed for them. To make their job easier, they will need higher level test case specification tools, or automated test case generation tools. Further work in web/VoIP test generation still needs investigation.

6. Conclusion

Converged applications deal with multiple protocols and multiple user interfaces. They bring a unique set of challenges to automated testing. In this paper, we have identified some general challenges and provided the starting point for a solution applied in the context of a mission-critical conferencing application. We described our automated test solution using open source testing tools — KitCAT which we developed internally, and HtmlUnit. Our testing approach has proved cost effective in multiple internal projects. We have documented our experience and have identified further areas for research in automated testing for converged applications.

Acknowledgments

We gratefully acknowledge the valuable contributions of our colleagues Gregory Bond, Hal Purdy, Thomas Smith and Pamela Zave.

References

[1] Apache Ant. http://ant.apache.org/.
[2] HtmlUnit. http://htmlunit.sourceforge.net/.
[3] JUnit. http://www.junit.org/.
[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, June 1999. IETF RFC 2616.
[5] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. 2002. IETF RFC 3261.
[6] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-time Applications. *RFC 1889*, Jan. 1996.
[7] SIP testing tools. http://www.cs.columbia.edu/sip/.
[8] SIPp. http://sipp.sourceforge.net/.
[9] SipUnit. http://www.cafesip.org/projects/sipunit/.
[10] V. Subramonian. KitCAT - A framework for Converged Application Testing. http://echarts.org/KitCAT/, 2008. Available from: `http://echarts.org/Downloads.html`.
[11] TTCN-3. http://www.ttcn-3.org/.
[12] XPath. http://www.w3.org/TR/xpath20/.