User Impersonation as a Service in End-to-End Testing

Boni García, Francisco Gortázar, Micael Gallego and Eduardo Jiménez
Universidad Rey Juan Carlos, Calle Tulipán S/N, 28933 Móstoles (Spain)

Keywords: End-to-End Testing, User Impersonation, Software as a Service, WebRTC.

Abstract: Testing large distributed heterogenous systems in cloud environments is a complex task. This situation becomes especially difficult when carrying out end-to-end tests, in which the whole system is exercised, typically through its graphical user interface (GUI) with impersonated users. These tests are typically expensive to write and time consuming to run. This paper contributes to the solution of this problem by proposing an open source framework called ElasTest, which can be seen as an elastic platform to carry out end-to-end testing for different types of applications, including web and mobile. In particular, this piece or research puts the accent on the capability to impersonate final users, presenting a real case study in which end-to-end tests have been carried out to assess the correctness of real-time communications among browsers using WebRTC.

1 INTRODUCTION

Modern software systems are increasingly complex. Nowadays, architectures involving distributed heterogenous services, cloud native, and microservices are more and more common. As usual, in order to accomplish a satisfactory level of quality for these systems, different aspects need to be addressed. First, the expectations of final users need to be met. Using the classical definition of Verification and Validation (V&V) by the distinguished professor of computer science Barry Boehm, this part is known as validation *-are we building the right product?-* (Boehm, 1979). Second, we need to ensure that the software meets its stated functional and non-functional requirements, i.e., its specification. This part is commonly known as verification *-are we building the product right?-*. Finally, we need to reduce the number of software defects (commonly known as bugs) in our system to the minimum, ideally to zero.

V&V include a wide array of activities, mainly divided in two groups. On the one hand, software testing (or simply testing) consists of observing a sample of executions (test cases), and giving a verdict on them. Hence, testing is an execution-based activity, and for this reason, it is sometimes called dynamic analysis. On the other hand, static analysis is a form of V&V that does not require execution of the software. Static analysis can work directly with the source code, and also with representation of the software, such as model of the specification of design. Common forms of static analysis include peer review or automated software analysis. Regarding the later, this technique is usually delivered as commercial or open source tools and services, commonly known as lint or linter.

This paper is focused in software testing, which is a broad term encompassing a wide spectrum of different concepts. Depending on the size of the System Under Test (SUT) and the scenario in which it is exercised, testing can be carried out at different levels. There is no universal classification for all the different testing levels. Nevertheless, the following levels are broadly accepted in the literature (García, 2017):

- Unit: individual program units are tested. Unit tests typically focus on the functionality of individual objects or methods.
- Integration: units are combined to create composite components. Integration tests focus on the interaction of different units.
- System: all of the components are integrated and the system is tested as a whole. There is a special type of system testing called end-to-end testing. In this approach, the final user is typically impersonated, that is, simulated using automation techniques.
- Acceptance: final users decide whether or not the system is ready to be deployed in the

García, B., Gortázar, F., Gallego, M. and Jiménez, E.
User Impersonation as a Service in End-to-End Testing.
DOI: 10.5220/0006752207070714
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 707-714
ISBN: 978-989-758-283-7

consumer environment. These tests can be seen as functional testing performed at system level by final users or customers.

The first three levels (unit, integration, and system) are typically carried out during the development phases of the software life cycle. These tests are typically performed by different roles of software engineers, i.e. programmers, testers, Quality Assurance (QA) team, etc. The objective of these tests is the verification of the system. On the other side, the fourth level (acceptance) is a type of user testing, in which potential or real users are usually involved (validation). As illustrated in Figure 1, the different tests levels are commonly depicted as a pyramid, in which the base is the unit tests (which in in theory are more numerous), while the number of other tests (integration, system, acceptance) is decreasing as long as we ascend to the top. This idea of a pyramid for the different testing levels was first proposed by Mike Cohn (Cohn, 2009).

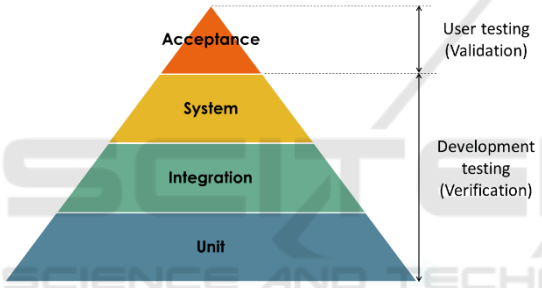

Figure 1: Testing levels and its relationship with V&V.

The capability to automate of the different tests levels has a direct impact on the project costs. Thus, user testing (acceptance) is unlikely to be fully automated, since the evaluation of the final consumer always comprises some kind of human intervention, and therefore this kind of tests can be costly. Development testing, on the other side, can and should be automated. Regarding top-level tests -system and end-to-end-, these tests typically drive an application through its user-interface, checking that the application returns the expected results. This approach works well in simple scenarios, but at the end of the day these tests are prone to potential problems, such as brittle logic, expensive to write, and time consuming to run (Fowler, 2012). This situation leads to the ice-cream cone anti-pattern, in which manual tests -which should be a reduced number on the top- increases its number more and more, while the number of down-level automated tests (integration and unit) is reduced (Scott, 2015).

This situation can become a real pain for software practitioners in the common case that the SUT is increasingly large and complex, such as distributed heterogenous, microservices, or cloud native systems. This kind of software systems aggregates many different distributes components, which are typically built and run applications based on Infrastructure as a Service (IaaS) combined with operation tools and services such as Continuous Integration (CI), container engines, or service orchestrators, to name a few.

This piece of this research contributes in the domain of end-to-end test automation (i.e. system tests in which the user is impersonated) for large complex distributed applications in cloud environments. To make easier this process for software practitioners, we have created an open source framework ElasTest. As we will discover, this framework provides advance test capabilities to ease the end-to-end tests process for different kind of applications, including web and mobile. Moreover, ElasTest provides the capability of impersonate final users on web and mobile devices, by extending the standard W3C WebDriver recommendation (Stewart, 2017). This service has been evolved into a fully Software as a Service (SaaS) model so that developers do not need to take into consideration problems related to computing resources scheduling, software provisioning or system scaling, providing a high-level test capability which can be referred as User Impersonation as a Service (UIaaS).

The remainder of this paper is structured as follows. Section 2 provides a brief overview in the state-of-the-art on end-to-end testing and user impersonation. Section 3 provides a description of the ElasTest framework. Then, section 4 provides extra details of the ElasTest's User Impersonation Service (called EUS in the ElasTest jargon). In order to validate our proposal, a case study has been performed using a videoconferencing web system built on the top of WebRTC. The description and results of this case study are contained in section 5. Finally, section 6 provides the conclusions, findings, and future work of this piece of research.

2 BACKGROUND

Testing distributed and heterogeneous software systems, running over interconnected mobile and cloud based platforms, is particularly challenging. To verify these systems, developers face with different problems, including the difficulty to test the system as a whole due to the number and

diversity of individual components, the difficulty to coordinate the test participants due to the distributed nature of the system, or the difficulty to test the components individually.

Recent research effort has tried to quantify the current state of the practice of the testing automation level for this kind of software systems. For instance, Lima and Faria have conducted an exploratory survey on testing distributed and heterogeneous systems that was responded by 147 software testing professionals that attended to two industry-oriented software testing conferences (Lima, 2016). The survey results confirm the existence of a significant gap between the current and the desired status of test automation for distributed heterogenous system, and confirm and prioritize the relevance of test automation features for these systems.

Many different contributions in the literature aimed to improve the current state of the art in end-to-end testing. For instance, the European Commission funded H2020 project TRIANGLE[1] is building a framework to help app developers and device manufacturers in the evolving 5G sector to test and benchmark new mobile applications in Europe. This framework evaluates Quality of Experience (QoE) and enable certification for new mobile applications and devices (Cattoni, 2016).

Regarding web and mobile applications, the main mechanisms used in the current state-of-the-art for the functional testing of these applications consists on impersonating a user through some kind of GUI automation technology, being Selenium[2] the most popular solution for web applications. In this domain, Selenium WebDriver is capable of drive automatically real browsers, such as Chrome, Firefox, Opera, Edge, Safari, etc., using different programming languages, such as Java, C#, Python, Ruby, PHP, Perl, or JavaScript. To that aim, Selenium WebDriver makes calls to the browser using each browser's native support for automation. The language bindings provided by Selenium WebDriver communicates with a browser-specific binary which acts as a bridge with the browser. The communication between the WebDriver script and the driver binary is done with JSON messages over HTTP using the so-called JSON Wire Protocol (Bruns, 2009). This mechanism, originally proposed by the Selenium team is being standardized in the W3C WebDriver recommendation (Stewart, 2017).

The second major component of the Selenium framework is called Selenium Grid. This component allows remote execution of Selenium WebDriver on distributed machines. The architecture of Selenium Grid is composed by a group of nodes, each running on different operating systems and with different browsers. Then, a central piece called hub (also known as Selenium Server) keeps a track of the nodes and proxies requests to them using JSON Wire Protocol/W3C WebDriver messages. This capability is used by the Appium[3] project to drive mobile devices. In Appium, instead of web browsers, mobile devices are registered in a central component called Appium Server. As depicted in Figure 2, following the Selenium Grid approach, the Appium Server is remotely controlled by means of Wire Protocol/W3C WebDriver messages, typically used by tests of scripts implementing the WebDriver API (Shah, 2014).

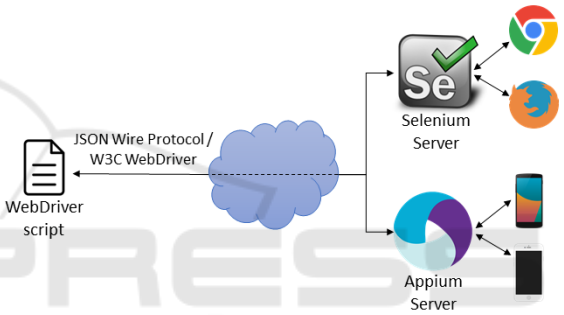

Figure 2: Selenium/Appium high-level architecture.

3 ELASTEST: AN ELASTIC PLATFORM TO EASE END-TO-END TESTING

ElasTest[4] is an open source[5] framework aimed to ease the end-to-end testing activities for different types of distributed applications and services, allowing developers and testers to assess their cloud applications in an elastic, and integrated environment. The proposed framework manages the full testing lifecycle, deploying and monitoring the SUT, executing the end-to-end tests and exposing the results to software engineers and testers.

In this paper we focus on the ElasTest capability to impersonate browsers and mobile devices. This service has been designed following a SaaS model in

[1] http://www.triangle-project.eu/
[2] http://www.seleniumhq.org/
[3] http://appium.io/
[4] http://elastest.io/
[5] https://github.com/elastest/

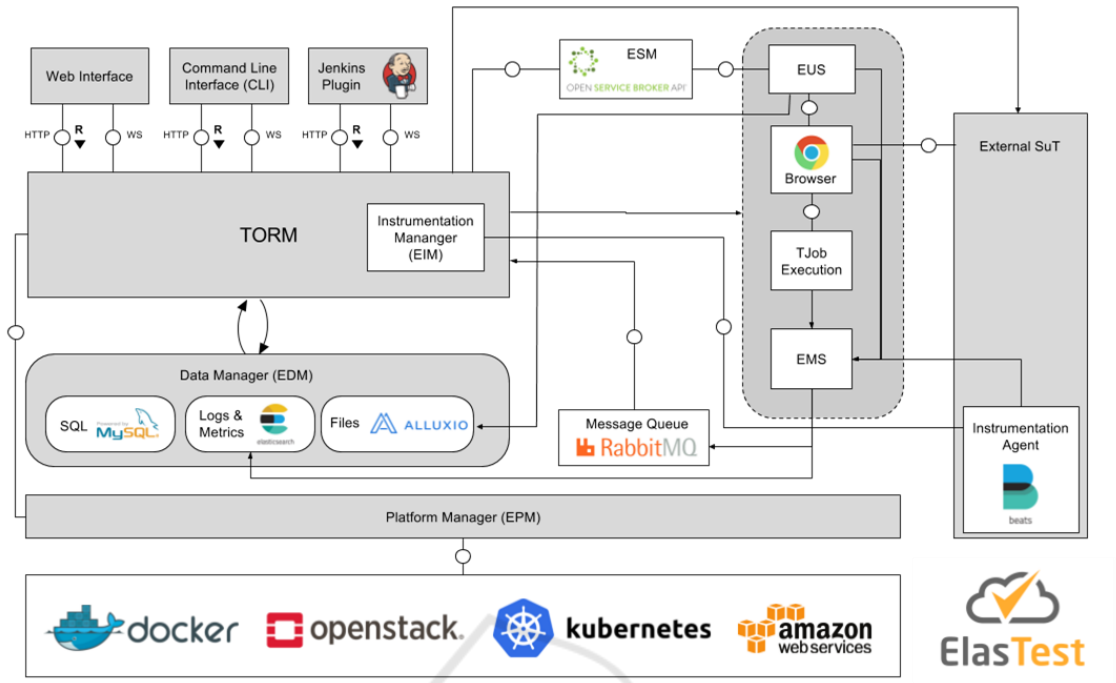

Figure 3: ElasTest architecture.

order to make transparent for the final user potential problems related to computing resources scheduling, software provisioning or system scaling.

In order to understand how EUS works, first we need to review the overall architecture of ElasTest, depicted in Figure 3. First of all, we find the ElasTest Test Orchestration and Recommendation Manager (ETM), which is the access point to the framework. It orchestrates all other components exposing different interfaces for consumers, such as a web GUI, a command line interface, and also an interface with a custom Jenkins plugin.

ElasTest follows a microservices approach, and the component which is responsible for discovering and operating the different services that ElasTest make available to tests is called ElasTest Service Manager (ESM). This component is based on the Open Service Broker API (OSBA)[6] for discovering, registering and unregistering services within the platform. RabbitMQ[7] is used as messaging queue for the events communication among the different services.

One of the key aspects handled out of the box by ElasTest is related with data management. During its operation, ElasTest gathers different sources of data from test execution, including SUT logs, different types metrics -including SUT resource consumption, packet-loss in the network traffic, or node failures, among others-, or custom files issued by services -e.g. browser/mobile session recordings carried out by EUS-. The component responsible for the persistence layer is called ElasTest Data Manager (EDM), and it has been built on the top of on MySQL[8] as relational database, Elasticsearch[9] as search engine, and Alluxio[10] as virtual distributed storage system.

The ElasTest Instrumentation Manager (EIM) provides the capability of instrumenting the SUT to inject potential system failures like packet-loss, network bandwidth adjustments to emulate real conditions, CPU bursting, and node failures, to name a few. To that aim, Beats[11] agents are installed together with the SUT.

Finally, the ElasTest Platform Manager (EPM) is the component responsible of isolating the ElasTest services from the underlying infrastructure. The supported cloud infrastructures are OpenStack[12],

[6] https://www.openservicebrokerapi.org/
[7] https://www.rabbitmq.com/

[8] https://www.mysql.com/
[9] https://www.elastic.co/
[10] https://www.alluxio.org/
[11] https://www.elastic.co/products/beats
[12] https://www.openstack.org/

Table 1: Extension to W3C WebDriver recommendation by ElasTest User Impersonation Service.

Method	Path	Description
POST	/session/{sessionId}/element/{elementId}/event	① Subscribe to a given event within an element
GET	/session/{sessionId}/event/{subscriptionId}	② Read the value of event for a given subscription
DELETE	/session/{sessionId}/event/{subscriptionId}	③ Remove a subscription
GET	/session/{sessionId}/vnc	④ Get remote session
DELETE	/session/{sessionId}/vnc	⑤ Delete remote session
POST	/session/{sessionId}/usermedia	⑥ Set user media for WebRTC
GET	/session/{sessionId}/stats	⑦ Read the WebRTC stats
POST	/session/{sessionId}/element/{elementId}/latency	⑧ Measure end-to-end latency of a WebRTC session
POST	/session/{sessionId}/element/{elementId}/quality	⑨Measure quality of a WebRTC session

Amazon Web Services[13] (AWS), Docker[14] and Kubernetes[15]. Moreover, Open Baton[16] is used for orchestrating the SUT and the network services within the ElasTest platform (Carella, 2015).

4 USER IMPERSONATION AS A SERVICE

As introduced in section 2, testers' need for user impersonation is more and more demanded. For that reason, nowadays there are several companies that are growing business models basing on exposing these capabilities through SaaS models, such as Saucelabs[17] or BrowserStack[18]. However, these solutions have relevant limitations. On the one hand, these services have very relevant costs, which limit their applicability for many projects. On the other hand, these services only impersonate the user from the perspective of its outgoing actions, but not from the perspective of its incoming perceived QoE.

ElasTest progresses beyond the current state of the art providing an advanced user impersonation as a service capability that provides GUI automation basing on open source paradigms and enables also the evaluation of the perceived quality of users on relevant scenarios such as real-time multimedia applications. As introduced in the section before, this feature has been implemented in the component called ElasTest User Impersonation Service (EUS), providing what we can call User Impersonation as a Service (UIaaS). This service is devoted to provide user impersonation for two types of user interfaces, i.e. web browsers and mobile devices.

In order to expose this capability through an API in a universal way, EUS has been implemented as an extension of the W3C WebDriver API. As presented in section 2, this recommendation is used to drive remote browsers and mobile devices, by means of a client-server technology implemented by Selenium and Appium respectively. The vision of EUS is to enhance the current support with additional advance capabilities in a seamless and integrated solution.

To that aim, EUS exposes a REST API based on JSON messages[19] which complements the W3C WebDriver specification. The definition to this REST API has been defined using Open API notation[20], and it is summarized in Table 1. In this table, the first operation allows to subscribe to events in a given element of the user interface. Then, second operation allows to read the value for a given subscription, and the third one allows to unsubscribe to that event. Operations 4 and 5 are related with the capability of remote GUI, provided by EUS out of the box by means of Virtual Network Computing (VNC). Using these commands, EUS allows to watch in real-time the use of a browser or mobile device, typically driven by a test script using the WebDriver API.

The last group of operations summarized in Table 1 are targeted for WebRTC applications. WebRTC is the umbrella term for a number of emerging technologies that extends the web browsing model to exchange real-time media with other browsers (Loreto, 2017). Market momentum around WebRTC

[13] https://aws.amazon.com/
[14] https://www.docker.com/
[15] https://kubernetes.io/
[16] https://openbaton.github.io/
[17] https://saucelabs.com/
[18] https://www.browserstack.com/
[19] http://elastest.io/docs/api/eus/
[20] https://www.openapis.org/

is growing very fast nowadays, and therefore, it is imperative for software testers to have a strategy in place in order to assess WebRTC applications efficiently. Nevertheless, testing WebRTC-based applications in a consistently automated fashion is a challenging problem. EUS contributes to the solution of this problem with proving advance features aimed to asses this kind of applications.

First of all, thanks to operation 6 presented in Table 1, the EUS is capable of faking the user media -video and/or audio- employed in a WebRTC communication with a custom video/audio file chosen by the tester. Then the operation 7, allows to read all the collection of WebRTC stats, which is a good indicator on Quality of Service (QoS) for WebRTC. These include traffic metrics such as network latency, network packet loss, network jitter, retransmissions, or consumed bandwidth.

Moreover, EUS enables to measure the end-user's perceived quality so that testing through the validation of the subjective perceived quality. To that, EUS analyses the multimedia QoE for audio and video using different full-reference algorithms, such as PESQ (Rix, 2001) for audio or SSIM (Wang, 2004) for video. Full-reference is type of QoE media-based algorithms, in which the degraded signal is compared with the original signal (Chikkerur, 2011). This, applied to EUS, means that a couple of browsers (or mobile devices) are needed, first one acting as media source and the other acting as media consumer. Internally, this process reuses the aforementioned publish-subscribe mechanism, in which quality events - audio or video- are published periodically.

Finally, EUS provides several extra capabilities in conjunction with the rest of ElasTest components. On the one hand, it records every session in a seekable recording, stored in EDM as a video file. This feature improves the traceability of tests, allowing users to check the evolution of test executions when required. On the other hand, EUS always gathers automatically the browser logs in every session. Again, this information is stored together the recording on EDM, and it can be a valuable source of information for developers and testers to try to trace the source of faults when tests are failing.

EUS has been implemented as a Spring Boot[21] REST service listening for the enhanced version of W3C WebDriver specification just presented. The EUS workflow starts with the invocation of a session creation carried out in a WebDriver script. Once the EUS controller receives this message (`POST /session`), proxies the message to a Selenium/Appium server provided on demand by EPM. Once the browser or mobile device is available, a unique session identifier (`sessionId`, which is always available in the operations described in Table 1) is sent as response. This parameter is used in successive requests to interact with the browser/mobile just created. At the end of the session, the script will invoke the termination command (`DELETE /session`) using the proper session identifier. At this point, the infrastructure resources are released by EPM, and the complete recording a logging data is sent to EDM.

5 CASE STUDY: TESTING WEBRTC APPLICATIONS MADE WITH OPENVIDU

In order to validate our proposal, a case study focused on WebRTC applications have been carried out. Concretely, we have cooperated with the team developing the project OpenVidu[22], an open source videoconferencing WebRTC framework. OpenVidu follows a client-server architecture and therefore is made up by two main components. On the client-side, the *OpenVidu Browser* is a JavaScript/ TypeScript library which allows to create video calls, join users to them, and send/receive media streams. On the server-side, the *OpenVidu Server* receives the operations from clients establishing and managing the video-calls.

In order to carry out end-to-end tests of WebRTC applications, it is mandatory to use browsers that implements the WebRTC stack, such as Chrome or Firefox. For that reason, in the OpenVidu project, end-to-end tests have been implemented using Selenium WebDriver. In the testing process carried out by the OpenVidu team, these tests were executed in a Jenkins Continuous Integration server. In this server the latest versions of Chrome and Firefox were installed, and Selenium sessions were executed through a virtual framebuffer display server -Xvfb-.

The research question driving this case study is the following: “Is the ElasTest user impersonation service capable of improving the end-to-end testing process within the OpenVidu project?”. To address this question, first an instance of ElasTest was

[21] https://projects.spring.io/spring-boot/

[22] http://openvidu.io/

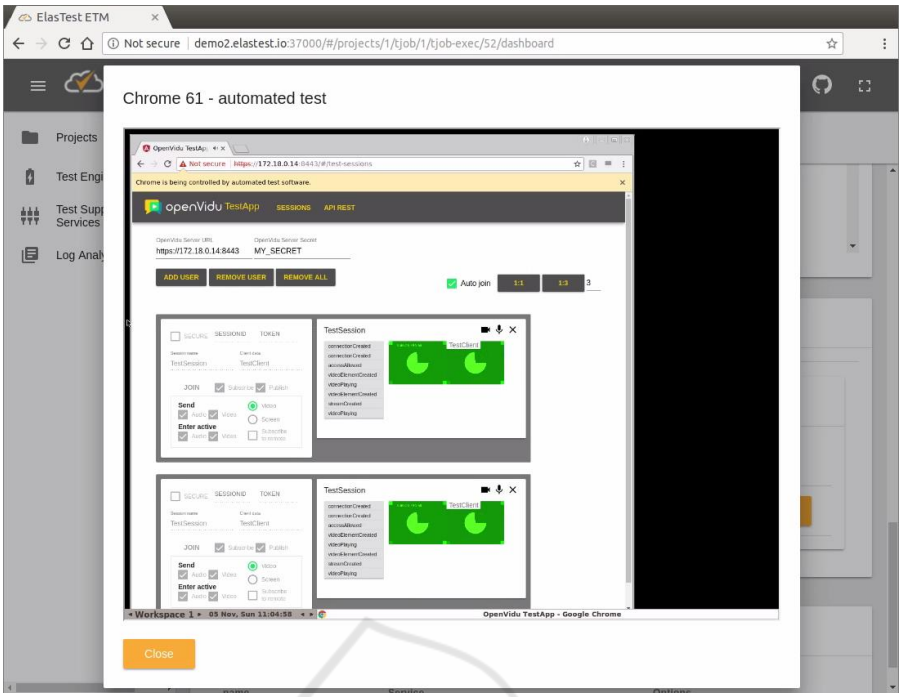

Figure 4: Screenshot of ETM/EUS during the execution of a OpenVidu end-to-end test.

provided to the OpenVidu team. The idea was to reuse the existing tests, adapting them to be executed inside ElasTest.

Due to the fact that the existing test suite was based on Selenium WebDriver, few changes was required in the test logic. The existing codebase was implemented in Java, and therefore the required change was related to the specific objects to control browsers -i.e. `ChromDriver` for Chrome and `FirefoxDriver` for Firefox-, by remote browser drivers, called `RemoteDriver` in Java. These objects require the URL to connect with the Selenium Server, which is implemented in ElasTest by EUS. When tests are executed inside ElasTest, this URL is available by reading the environmental variable `ET_EUS_API`. The source code of these tests is available on GitHub[23].

The SUT lifecycle was managed by ElasTest together with the test execution. In this case study, a Docker Compose[24] script was configured within the ETM, defining the OpenVidu application under test and its dependencies. Figure 4 provides an ETM screenshot of the exectution of one end-to-end test against the SUT while it is executed by the EUS. As explained in the section before, once the test finished, a recording of the session navigation, together the the browser logs is stored persistenly in ElasTest.

Once the tests were adapted and executed in ElasTest, we were able to draw some conclusions about the UIaaS. First of all, we conclude that the fact that EUS is based on the W3C WebDriver standard, facilitates its adoption in an existing test codebase. Second, the capability to provide different types of browsers and version in a semanless and elastic manner is very valuable for testers, since it avoids to manage directly the infrastructure reducing the efforts required mainly in DevOps side, and providing valuable assets to create compatibity tests for testers. Finally, the capability for storing to the browser session recording and logging makes a big difference for OpenVidu testers. This feature allows to trace and debug failed tests in a much more realiable way than before, in which testers were blind to trace errors of tests executions on their Jenkins infrastructure.

6 CONCLUSIONS AND FUTURE WORK

Software testing is the most commonly performed activity within V&V. Modern web and mobile applications are characterized by rapid development

[23] https://github.com/elastest/demo-projects
[24] https://docs.docker.com/compose/

cycles, which supposes that testers tend to pay scant attention to automated end-to-end test suites. As a result, this kind of tests is usually abandoned or poorly performed.

This paper introduces ElasTest, an open source platform aimed to ease end-to-end tests for heterogenous large distributed systems. The mision of ElasTest is to make easier the developers' life. To that aim, among other capabilities, ElasTest implements what we can call User Impersonation as a Service (UIaaS). This service enables the impersonation of end-users' in their tests through GUI instrumentation. This service provides full compatibility with external browser/mobile drivers, but enhanced with extra capabilities, such as event subscription, log gathering, or advance media capabilities for WebRTC applications. This service have be built extending the W3C WebDriver specification, and therefore, popular technologies such as Selenium and Appium are completely compatible with ElasTest.

At the moment of this writing, ElasTest is still in its infancy. Therefore, some features are still under development. For instance, the measurent of the end-users' perceived QoE is still ongoing. Measuring QoE is in general a complex topic and this task shall perform the appropriate research activities for evaluating the most suitable way of doing it, which may involve simple mechanisms such as evaluation of response-time from the GUI.

ACKNOWLEDGEMENTS

This work has been supported by the European Commission under projects NUBOMEDIA (FP7-ICT-2013-1.6, GA-610576), and ElasTest (H2020-ICT-10-2016, GA-731535); by the Regional Government of Madrid (CM) under project Cloud4BigData (S2013/ICE-2894) cofunded by FSE & FEDER; and Spanish Government under project LERNIM (RTC-2016-4674-7) cofunded by the Ministry of Economy and Competitiveness, FEDER & AEI.

REFERENCES

Boehm, B.W., 1979. Software engineering: R&D trends and defense needs. *Research directions in software technology*, *1*, p.977.

Bruns, A., Kornstadt, A. and Wichmann, D., 2009. Web application tests with selenium. *IEEE software*, *26*(5).

Carella, G.A. and Magedanz, T., 2015. Open baton: A framework for virtual network function management and orchestration for emerging software-based 5g networks. *Newsletter*, *2016*.

Cattoni, A.F., Madueño, G.C., Dieudonne, M., Merino, P., Zayas, A.D., Salmeron, A., Carlier, F., Saint Germain, B., Morris, D., Figueiredo, R. and Caffrey, J., 2016, June. An end-to-end testing ecosystem for 5G. In *Networks and Communications (EuCNC), 2016 European Conference on*(pp. 307-312). IEEE.

Chikkerur, S., Sundaram, V., Reisslein, M. and Karam, L.J., 2011. Objective video quality assessment methods: A classification, review, and performance comparison. *IEEE transactions on broadcasting*, *57*(2), pp.165-182.

Cohn, M., 2009. The forgotten layer of the test automation pyramid. *Mike Cohn's Blog–Succeeding with Agile*, Accessed on November 2017. *http://blog. mountaingoatsoftware. com/the-forgotten-layerof-the-test-automation-pyramid*

Fowler, M., 2012. Test pyramid, Accessed on November 2017 *https://martinfowler.com/bliki/TestPyramid.html*

García, B., 2017. *Mastering Software Testing with JUnit 5*, Packt Publishing. Birmingham.

Lima, B. and Faria, J.P., 2016, July. A Survey on Testing Distributed and Heterogeneous Systems: The State of the Practice. In *International Conference on Software Technologies* (pp. 88-107). Springer, Cham.

Loreto, S. and Romano, S.P., 2017. How Far are We from WebRTC-1.0? An Update on Standards and a Look at What's Next. *IEEE Communications Magazine*.

Rix, A.W., Beerends, J.G., Hollier, M.P. and Hekstra, A.P., 2001. Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on* (Vol. 2, pp. 749-752). IEEE.

Scott, A., 2015. Introducing the software testing ice-cream cone (anti-pattern). Accessed on November 2017. *https://watirmelon.blog/2012/01/31/introducing-the-software-testing-ice-cream-cone/*

Shah, G., Shah, P. and Muchhala, R., 2014. Software testing automation using Appium. *International Journal of Current Engineering and Technology*, *4*(5), pp.3528-3531.

Stewart, S. and Burns, D., 2017. WebDriver. *Working draft, W3C*.

Wang, Z., Bovik, A.C., Sheikh, H.R. and Simoncelli, E.P., 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, *13*(4), pp.600-612.