Integrated TaaS Platform for Mobile Development: Architecture Solutions

Oleksii Starov and Sergiy Vilkomir
Department of Computer Science
East Carolina University
NC, USA
{starovo12, vilkomirs}@ecu.edu

***Abstract*—This paper examines the Testing-as-a-Service (TaaS) solutions in mobile development and proposes a universal TaaS platform: Cloud Testing of Mobile Systems (CTOMS). CTOMS is an integrated solution with a core infrastructure that enables the scaling of additional functionalities. The CTOMS's benefits are explained, the architecture of the system is described in detail, and technical solutions are listed based on the feasibility study that resulted in creation of the first version of CTOMS for Android development.**

***Index Terms*—Testing-as-a-Service (TaaS), mobile application development, Android, integrated solution.**

I. Introduction

Mobile development is a special case of general software development with its own challenges and features. Mobile application testing has some similarities to website testing as both involve validation in many environments (smartphones and browsers, respectively). The general requirements for both types of testing are similar—applications should function correctly, efficiently, and be reliable and secure in all environments. However, mobile testing presents new activities and requires more effort because it includes web applications that work within mobile browsers or hybrid variants wrapped in native code [1]. It also involves a large number of possible combinations of mobile devices and operating systems (OS) [2]. Finally, mobile testing involves the use of actual hardware and so testers need additional knowledge and skills such as build installation or crash-log retrieving.

Good quality testing is necessitated by the increase in mobile applications for critical daily activities [3–7]. To facilitate the testing of mobile applications, various cloud benefits are used and different TaaS or supporting systems already exist [8–13]. These services cover all specific mobile testing needs. The underlying systems are very complex and offered as cloud services because small developer teams often cannot afford them as internal solutions.

This paper describes an integrated TaaS solution CTOMS with scalable architecture. All core testing functionalities are jointly implemented in one platform over the cloud. This provides the possibility of different use cases and the ease of adding new functionalities, such as non-functional testing or test planning approaches. The implementation of this single system should be less complicated than for separate systems for each testing activity.

This paper is organized as follows. Section II describes the state-of-the-art in cloud and mobile application testing. Section III provides the requirements for TaaS for mobile development and describes the structure of an integrated solution as well as the high-level architecture of the CTOMS platform. Sections IV and V discuss architectural components of CTOMS in detail. Section VI provides the results of the feasibility study and describes the implementation of the initial version of CTOMS for Android development. Finally, Section VII presents the summary and future plans.

II. Related Works

Testing as a service over cloud is a highly topical issue at the moment [14-17] that includes testing services for mobile development [18]. Cloud testing technical questions are analyzed in [19]. General questions about cloud architecture are discussed in [20, 21], including the creation of a cloud testing framework with plug-in architecture [22].

The developers of mobile applications are using several kinds of TaaS systems or supporting cloud services: cloud of devices for actual testing on different models and configurations [8–10], services for build distribution and test team management [11–13], and services for test planning [23]. General online static code analyzers or specific model checking approaches can also be included. At the same time, necessity for other testing options still exists, such as testing a new device model or an update to an OS against popular or legacy applications. For example, a service exists to test a new smartphone against top apps in the market [24].

The situation described here reflects the main differential challenges of mobile development [1]: support of many hardware and software platforms [2], correct work with a variety of sensors, interconnections with other applications, and high requirements for user experience (i.e., the quality of the user interface). To meet these challenges, many of TaaS systems serve as a "cloud of devices" that provides a wide range of remote smartphones for manual or automated testing. Automated tests are usually GUI-based.

Two research attempts within universities to create and investigate test-bed cloud solutions for mobile development are

Accepted for publication by IEEE. © 2013 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/ republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

the Android Tactical Application Assessment and Knowledge (ATAACK) Cloud [25] and SmartLab [26]. Both are distributed systems that connect a set of mobile devices under the Android OS for application investigation, development, and testing.

The popularity of reliable mobile applications and the spread of critical mobile applications such as in [3–7] require the use of additional testing techniques to ensure quality, security, and performance, for instance, application of test planning techniques to guarantee satisfactory coverage of software-hardware configurations [23], a service for long-term "monkey" testing of applications [10], and embedded in emulator approaches to check performance [27].

Several studies have dealt with the separate technical questions such as GUI-based testing automation [28], security testing of mobile applications [29], and usage of Hadoop for tests distribution [19], among others. These solutions represent a wide range of different approaches (e.g., dynamic testing and static analysis), and all of them can be leveraged into the CTOMS platform.

Android was chosen to be a target mobile platform for the first version of CTOMS implementation because (1) it is the most popular and widespread mobile OS, according to several statistics agencies [30]; (2) the problem of testing apps on a big set of heterogeneous devices and configurations is more evident for Android [2]; (3) Android is an open-source platform with a developed bug-tracking system [31] that can support investigations and analyses; and (4) Android aims to meet enterprise requirements [32].

At the same time, software engineering activities, the testing process, and even life cycles of mobile applications are similar for development under Android, iOS, Windows Phone, or other systems [33].

III. TaaS platform for Mobile Development

A. General Requirements

This paper attempts to provide a comprehensive view of the TaaS platform for mobile development. From this point onward, we will consider TaaS for mobile development as a cloud service that provides the following functionalities:

- Testing on heterogeneous software-hardware configurations.
- Embedded application of functional and non-functional testing techniques, including static analysis.
- Usage of statistics as a possible basis for testing techniques and reliability evaluations.
- Test planning and management facilities, including test team operation.
- Multidirectional testing (i.e., testing under different roles: app developer, device, or OS producer).
- Multitenancy and user management.
- Possible inclusion in continuous integration processes or usage through Application Programming Interfaces (APIs).

The last two requirements should ease the use of such systems in more popular ways, especially among mobile developers. Different scenarios may require automated usage, a private or public cloud, or automation with a build process, and other requirements.

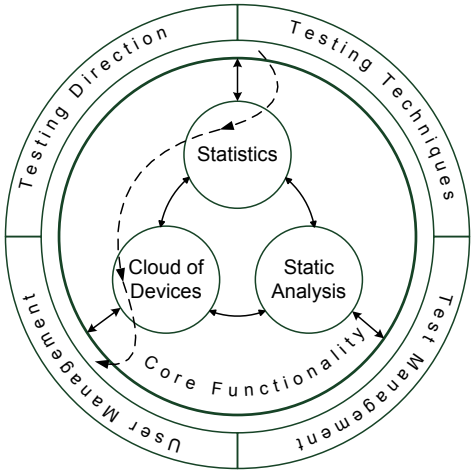

Fig. 1. Structure of an integrated TaaS platform

The consideration of the different TaaS functionalities according to these requirements leads to the idea of a mutually beneficial integrated solution. Figure 1 shows that the proposed core functionality of such a solution will have three main components: a cloud of devices, a static analysis engine, and a statistics sub-system. The desired high-level testing services will use these components separately or jointly when performing their functions. For instance, the dashed line in Figure 1 shows a scenario in which, based on statistics, a particular set of devices is chosen to perform automated testing in the cloud. The cloud of devices can be used for functional or manual testing after the application of some testing techniques to select the proper device coverage or can be used for non-functional testing such as the long-term stress testing of the app on selected devices. Testing techniques can use information provided by statistics to calculate coverage, or statistics can be used to determine the duration of reliability testing. Static analysis can be applied at any time simultaneously. Finally, a cloud of devices with supporting databases (data storages) can be used to test in different directions (as will be shown in detail later in relation to the CTOMS master application).

Figure 1 shows four groups of high-level testing services built onto a core infrastructure. Thus "testing techniques" means performing functional tests on a set of remote devices such as unit-tests or GUI-based test scripts [29] with or without the application of supporting testing techniques, e.g. combinatorial testing to calculate desired configurations coverage [34]. Non-functional testing is also taken into account and involves the application of performance testing on remote devices using frame rate counters [27], statistical testing, or security testing by conducting static analysis of source codes [28].

"Test management" means the presence of supporting services such as build distribution [10–13], test plan creation, and testing team management.

"User management" describes the need to separate the users of the system and provide access rules. This includes general user management and global multitenancy requirements.

"Testing direction" means the availability of different perspectives on testing. For instance, a user may utilize a service both as an app developer to test apps on devices and as a hardware or OS developer to test a new device (connected to the cloud on his/her side) or the OS on this device against available binaries and test artifacts.

B. High-Level Architecture

The high-level architecture of the proposed CTOMS platform as a cloud testing service satisfies the requirements of an integrated TaaS solution as mentioned above. From an architectural point of view, CTOMS is a distributed system that can be implemented in different ways. Architecture depends mainly on the organization of a dominant "cloud of devices" core feature. Other core sub-systems (e.g., static analysis and statistics) are simpler web applications that do not require special load balancing or synchronization of data but, of course, can be deployed to some cloud infrastructures. The following list presents the variants of CTOMS's possible implementation from a cloud-less to a cloud-full solution:

1) *Single server*. This system consists of the one web application that operates connected devices and/or emulators, manages a local database, and provides extended web interface to end-users. Devices can be connected by TCP (e.g., Wi-Fi) or through USB. Of course, such a solution is not scalable and has crucial limitations on the number of operated devices.
2) *Several servers*. This system represents master-slave architecture, i.e., one server serves as the master that provides a web interface to end-users. Others serve as slaves or leaf nodes that operate devices (henceforth, terminology "node" will be used). Node computers perform a series of tests, gather statistics, and provide the results to the master. Such a solution does not use the power of cloud computing to increase efficiency of testing. For instance, it will be slow just in case many requests are made at the same time.
3) *Master application in the cloud.* This solution is similar to the previous one, except the master application is deployed to some cloud. In this case, the system leverages the power of auto-balancing for master application and auto-replication for its data storage. Usage of adequate algorithms for test distribution should guarantee a satisfactory speed for the service.
4) *Cloud infrastructure.* This distributed system is built from the infrastructure level with the aim of providing desired functionality. In other words, algorithms to distribute tests (i.e., load balancing of the test execution layer) are integrated into the infrastructure. This is the highest level of cloud nature that should guarantee the highest efficiency of work.

This paper advocates the third variant because it has cloud nature with perceived adequate efficiency and is feasible for implementation by a small research team.

Figure 2 illustrates the architecture of CTOMS according to the third level of distributed system implementation. This approach separates the presentation and the platform and the information logical layers of the cloud solution, which is similar to the notation used in [20].

The platform layer is represented by the master application, e.g., on Google App Engine (GAE) cloud [35], and optional Hadoop instances layer that leverages the MapReduce algorithm to distribute a test between nodes and gather the results. Hadoop instances can be organized on other clouds (e.g., Amazon EC2). The master application provides the presentation to the end-user and to slave node computers.

The information layer is represented by the cloud data storage. We use notion data storage instead of a database to highlight that a system also needs to efficiently keep files (binaries, source codes, test scripts, screenshots, etc.), so some cloud storage can be used.

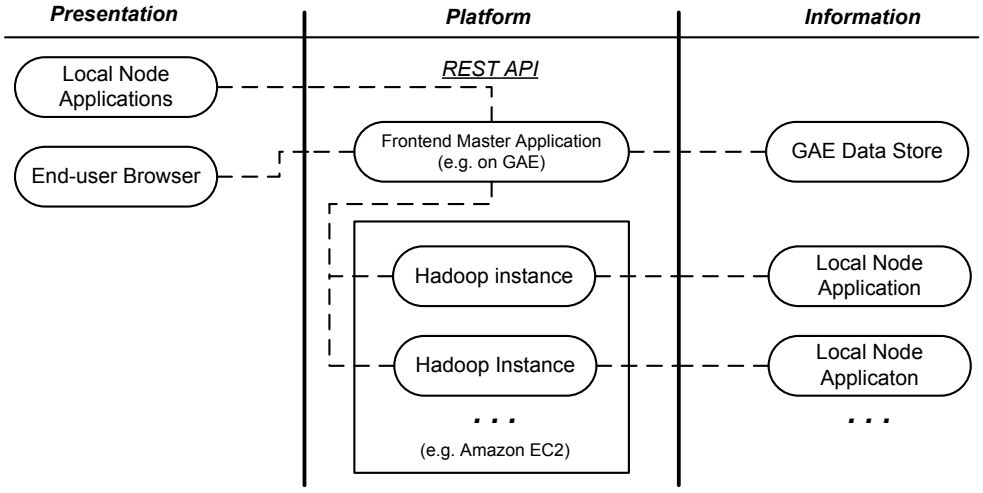

Fig. 2. Architecture of CTOMS platform as a cloud system

The current CTOMS implementation is developed entirely according to Fig. 2, except for the Hadoop layer. The master application is deployed on GAE. Interactions between all system components are provided through web services. RESTFul web services are suggested due to their lightweight features and popularity as APIs among mobile developers. Sections IV and V describe the detailed architecture of node and master applications, respectively.

IV. Architecture of a Node Application

Node application serves as a slave server application. It is supposed to be installed on the contributor's site to operate connected devices. The main responsibilities of a node are managing devices, performing tests, and general settings of interconnections with the master.

Figure 3 illustrates the current architecture of the CTOMS node application. The figure shows that the node provides two interfaces: a web interface to the local user and a web services tier to communicate with the master application. The first interface is used for settings, device information adjustments, and local application testing to check the system's serviceability. Corresponding sub-systems are discussed below.

The device manager uses SDK through the device driver to retrieve information about connected devices. The present CTOMS version is targeted at Android testing only, so Android SDK [36] is used, but the architecture is universal and specific components can be substituted (e.g., to update the current structure for the Windows Phone platform, another SDK can be used during system implementation). The user (administrator of particular node computer) adjusts the device information (i.e., vendor, size of memory, type of Internet connection, etc.) to send to the master application. At the same time, the user can choose active devices for local test sessions. The device manager keeps all information in the internal (embedded) database.

The testing manager uses the testing driver to operate the testing tools, e.g., tools for GUI test automation using scripts [37]). The testing manager consists of two supporting components: test preparation utility and test queue. The first component is used to adjust the test scripts or test commands, (e.g., to parallelize them to run simultaneously on several connected devices; such an approach is implemented in the current CTOMS version for MonkeyRunner scripts [37]). The second component is used to establish the order of performed test sessions. The testing process uses a local file system to prepare test scripts and save temporary screenshots, etc.

The settings manager operates with general settings: the path to SDK, the URL address of the master application, etc.

Figure 3 shows that the architecture of a CTOMS node has a cross structure based on total reuse of the internal components. This means that the same functions can be performed during local testing by the user-administrator or by a request from the master application.

Additional attention must be given to the web services tier. Two variants of the organization of the interconnections with the master application are possible:

- Web services are exposed through a public IP address. This means that the node must be accessible through the Internet.
- The logic of universal periodical calls from node application to master must be determined.

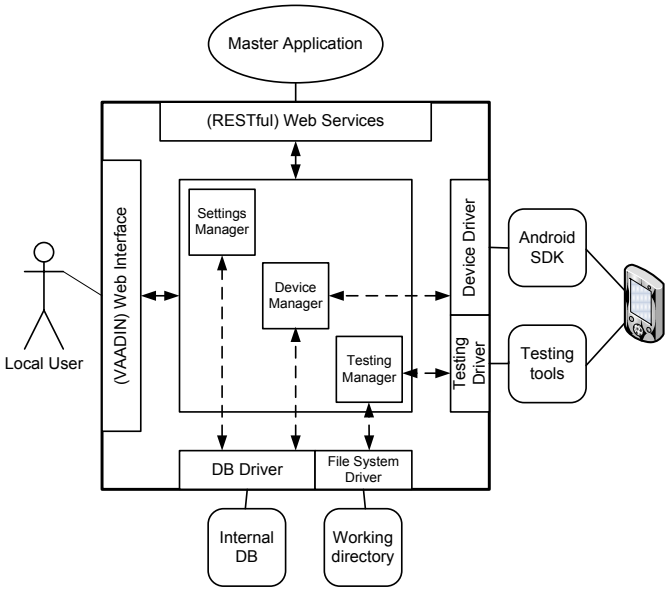

Fig. 3. Architecture of a CTOMS node

Each of the variants has pros and cons. While the first one is usually complicated to establish, the second one increases Internet traffic and the load on the master application and decreases the service speed. The current CTOMS solution uses the first variant because of economic reasons. The GAE offers a free service within limited usage of the cloud that includes quotes on the number of http requests. Thus the second solution could be expensive in the case of multi-nodes and long-term usage. A supporting approach such as localtunnel [38] is widely used during development to expose the local web server to the Internet.

V. Architecture of the Master Application

The master application of the CTOMS system aims to organize the core functionality of the integrated TaaS solution. It collects information about the whole system, organizes the work of nodes, and gathers testing results from them. These functionalities represent the "cloud of devices" core sub-system. In addition, the master application keeps statistics and artifacts in cloud data storage and can contain static analysis facilities. The architecture discussed in this section does not reflect static analysis because it requires another fundamental study, but instead shows the integration of functional testing techniques (i.e., one of the higher-level layers). Figure 4 shows the architecture of the current CTOMS master application. The architecture like the CTOMS node contains two interfaces (presentation layers): web interface to end-user and web services to communicate with slave nodes. The proposed minimal list of web services should contain the following groups:

- *Nodes info*. Service that node application invokes to send information about available devices and general settings such as password for node.
- *Load check*. Service that master application invokes from node to get information about test queue (i.e., load on the slave server).
- *Testing process*. Set of services to send testing task (binary, test script, list of devices, etc.), to get test results and upload/download artifacts.
- *External APIs*. These serve as interfaces to external tools (services), which provide calculations for some testing techniques. For instance, the current CTOMS implementation uses a NIST ACTS tool wrapped into APIs to build the coverage of configurations.

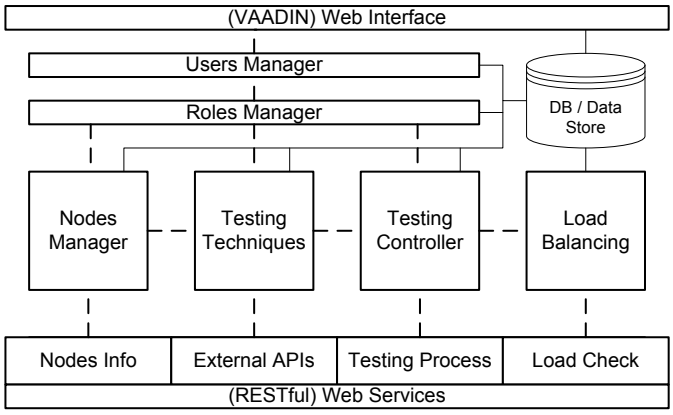

Fig. 4. Architecture of the CTOMS master

As for the number of web services groups, four main components exist in the system: nodes manager, testing techniques provider, testing controller, and load balancing controller. Each of these uses cloud data storage (database), but mostly the testing controller because it manages many test artifacts. Users manager and roles manager layers are above the working components and provide service settings depending on the current user and his/her chosen role (perspective).

The next issue is load balancing. For its simplest variant, the following greedy algorithm is proposed and is used in the current CTOMS version. The system operates with a list of sets of alternative devices. If the user manually chooses an exact group of devices to test on, each item on the list will be a set with one exact device, except situations when several identical devices are connected to the platform. If the user applies embedded testing techniques (to calculate coverage), each set in the list will contain devices that satisfy some set of parameters (i.e., screen resolution, processor, etc.). Then the algorithm chooses devices on the nodes with the smallest current queue of tasks.

VI. Feasibility Study

This section lists technical notes on actual CTOMS implementation more comprehensively. The available version of CTOMS was developed as a feasibility study with the following goals:

- To meet technical risks given the resource-constrained research team.
- To create a demonstrative version of the main concepts.
- To obtain a platform that suffices for first case studies with the flexibility to be extended and/or modernized.

Figure 5 illustrates the created system and points out that the end-user can operate both the node application on the local computer and the master application deployed in the cloud. The node application provides use cases more typical to the contributors of devices. The master application provides the final desired functionality of the system for mobile app developers. The possible role of device/OS developer is more complex and requires the user to connect the target device to node and to organize its testing through the master. The following key variants of usage are implemented.

Mobile apps automate testing using MonkeyRunner test scripts. MonkeyRunner is the standard tool of Android SDK that provides automated execution of GUI-based tests written in Python [37]. A user in the role of a mobile developer uploads binary and test files, which results in a set of screenshots taken from different devices for comparison. This is available on both master and node applications and can be used for functional and user interface testing.

Application of combinatorial testing techniques. The master application provides functionality to test mobile applications according to selected coverage of configurations criterion: base choice, pair-wise testing, etc. Under these configurations, the system understands the possible combinations of parameters such as device producer, version of Android OS, screen resolution, type of Internet connection, and memory size, etc.

Multidirectional testing. In the master application, the user can specify him/herself as a device or OS developer and check one of the connected devices (or the OS installed on this device) against chosen applications. In the results, the user receives screenshots for comparison with information collected during previous testing these applications (i.e. with oracle screenshots that are marked as "correct" in the system).

The CTOMS node was developed using Java web technologies as a self-executable war file with an embedded Tomcat server and Derby database. It can work on any local computer with JVM and Android SDK installed. The process of node application installation is easy and requires only copying the war file in a desired directory and running the console command. At the same time, a web application can be deployed on any other Java servlet container with a more detailed process for configuration.

The node application provides just three tabs in its web interface: settings, device manager, and local testing. The user (administrator of node computer) can provide the settings to specify the SDK location, the URL of the master application to use, and the secured node password. It is possible to manage information about connected devices and to perform local testing sessions to check the correctness of the installation.

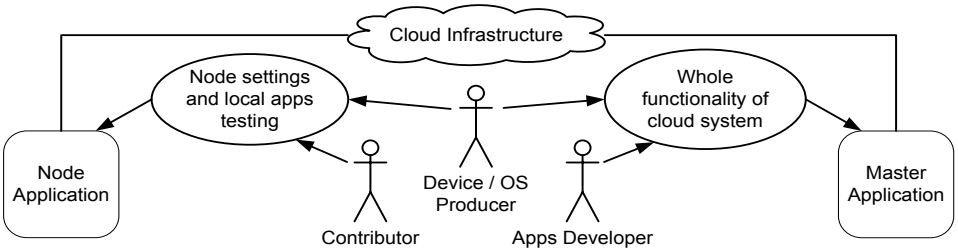

Fig. 5. First CTOMS platform implementation

The current version of the node application uses the "adb" command line tool (Android Debug Bridge) from SDK to retrieve information about connected devices and the MonkeyRunner tool for testing. Test scripts are assumed to be real test scripts used by a development team on its own site, but aimed for screenshot-based testing. This means the script should use standard MonkeyRunner facilities to save screenshots for desired checkpoints and that the user-tester will have the ability to compare screenshots taken from different models. Technical notes on the CTOMS node implementation are as follows:

- The VAADIN framework [39] was used to implement a whole user web interface because it provides the rapid development of modern web applications (not websites) as desktop ones.
- Jersey JAX-RS reference implementation was used to expose RESTful web services [40].
- Test script parallelization was implemented to update initial scripts for simultaneously performing on several connected devices.
- The user is provided with suggestions about screenshot similarities. A comparison is implemented as 90% matching using MonkeyRunner facilities. At the same time, the user can update the pass/fail statistics (i.e., specify correct and wrong screenshots that will be sent to the master application and then possibly used for device testing against this app).
- The testing queue organizes the sequential execution of the different testing sessions within web server and is implemented using Hazelcast distributed lock [41].

The master is implemented as a Google App Engine Java web application. A user interface is also developed using the popular VAADIN framework, and the RESTful web services tier uses Jersey implementation. File storing is implemented through GAE Blobstore [35], and usage of cloud data storage [35] is considered as an alternative variant for future versions.

The master application provides the user with the list of all devices connected to the system and organizes the testing of the same application on different nodes, if needed. An initial load balancing approach is used, i.e., the selection of the freest nodes that have the desired model connected.

The master application adds additional functionality for mobile app developers to apply combinatorial techniques for testing. The user can choose a set of models to test on, not manually, but using, for instance, a pair-wise approach. An example of its application for different Android configurations can be found at [34]. To calculate the required test cases, CTOMS uses the ACTS tool provided by NIST. The developed system includes a separate server that exposes ACTS functionality as RESTful APIs.

Another new functionality of the master application is "device developer" perspective. The user can choose "node server" from the list, provide a password to it, and select the device he/she wants to check. Then the user can choose apps from the overall list (last public test sessions) and run test cases used for their testing on the selected device. No additional functionality from the CTOMS node is needed, just the organization of the scenario on the CTOMS master.

VII. Conclusion and Future Work

This paper introduces an integrated solution of TaaS for mobile development. The CTOMS platform was presented as the first implementation of such a solution to demonstrate its benefits and new features. Mainly based on the cloud system that connects different mobile devices, CTOMS provides the functionality for testing mobile applications, applying testing coverage techniques, and testing connected devices against apps.

The high-level architecture of TaaS solution and the detailed architecture of the CTOMS platform were described. The implemented system serves as feasibility study that proves the possibility of similar investigations within small and resources-constrained research groups. Further work has two main directions: extension of CTOMS with new features, especially with approaches for performance and security testing, and case studies using the platform. For instance, we are interested in obtaining experimental evidence of a correlation between OS updates and the appearances of defects in older apps. We will also support MonkeyRunner test scripts enhanced by AndroidViewClient [42] and ViewServer [43] solutions.

References

[1] A. Wasserman, "Software engineering issues for mobile application development," *Proceedings of the Workshop on Future of Software Engineering Research* (FoSER 2010), at the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Santa Fe, USA, November 7–11, 2010, pp. 397–400.

[2] uTest, Inc., The Essential Guide to Mobile App Testing (free eBook). Available at: http://www.utest.com/landing-blog/essential-guide-mobile-app-testing, accessed Feb. 2013.

[3] Bank of America, "Mobile Banking," https://www.bankofamerica.com/online-banking/mobile.go, accessed Feb. 2013.

[4] K. Moser, "Improving Work Processes for Nuclear Plants", in *American Nuclear Society Utility Working Conference*, Hollywood, Florida, USA, August 5–8, 2012.

[5] D. B. Work, and A. M. Bayen, "Impacts of the Mobile Internet on Transportation Cyberphysical Systems: Traffic Monitoring using Smartphones," in *Proceedings of National Workshop for Research on High-Confidence Transportation Cyber-Physical Systems: Automotive, Aviation and Rail*, 2008.

[6] P. Leijdekkers, and V. Gay, "Personal Heart Monitoring and Rehabilitation System using Smart Phones," in *International Conference on Mobile Business*, Citeseer, 2006, p. 29.

[7] D. F. Carr, "Hurricane Sandy: Mobile, Social Tools Help Emergency Management," Brainyardnews, 2012. Available at: http://www.informationweek.com/thebrainyard/news/social_media_monitoring/240012463/hurricane-sandy-mobile-social-tools-help-emergency-management.

[8] Perfecto Mobile, "The MobileCloud Company," http://www.perfectomobile.com/, accessed Feb. 2013.

[9] Keynote DeviceAnywhere, "The Mobile Testing Platform," http://www.keynotedeviceanywhere.com, accessed Feb. 2013.

[10] "Apkudo," http://www.apkudo.com/, accessed Feb. 2013.

[11] "TestFlight," https://testflightapp.com/, accessed Feb. 2013.

[12] "Air On App," http://www.aironapp.com/, accessed Feb. 2013.

[13] "Launchpad," http://launchpadapp.com/, accessed Feb. 2013.

[14] K. Inçki, I. Ari, and H. Sozer, "A Survey of Software Testing in the Cloud," in *Proceedings of 2012 IEEE Sixth International Conference on Software Security and Reliability Companion*, 2012, pp. 18–23.

[15] S. Vilkomir, "Cloud Testing: A State-of-the-Art Review," V. Kharchenko, Ed. Information & Security: An International Journal, Volume 28, Issue 2, Number 17, 2012, pp. 213–222.

[16] S. Tilley, and T. Parveen, "Software Testing in the Cloud: Perspectives on an Emerging Discipline," Information Science Reference, November 2012.

[17] W. Tsai, X. Chen, L. Liu, Y. Zhao, L. Tang, and W. Zhao, "Testing as a service over cloud," in *Fifth IEEE International Symposium on Service Oriented System Engineering*, 2010.

[18] Priyanka, I. Chana, and A. Rana, "Empirical evaluation of cloud-based testing techniques: a systematic review," ACM SIGSOFT Software Engineering Notes archive, Volume 37, Issue 3 (May 2012), ACM New York, NY, USA, 2012, pp. 1–9.

[19] S. Tilley, and T. Parveen, Software Testing in the Cloud Migration & Execution, Springer, 2012.

[20] J. Rhoton, and R. Haukioja, Cloud Computing Architected: Solution Design Handbook, Recursive, Limited, 2011.

[21] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, Distributed Systems: Concepts and Design, Addison-Wesley, 5th ed., 2011.

[22] W. Jenkins, S. Vilkomir, P. Sharma, and G. Pirocanac, "Framework for Testing Cloud Platforms and Infrastructures, " in *Proceedings of the CSC 2011, International Conference on Cloud and Service Computing*, December 2011, pp. 134–140.

[23] "Keynote DeviceAnywhere Test Planner," http://www.keynotedeviceanywhere.com/mobile-test-planner.html, accessed Feb. 2013.

[24] D. Rowinski, "Mobile Carriers and OEMs Get Android App Testing Cloud from Apkudo," ReadWrite, February 2012. Available at: http://readwrite.com/2012/02/07/mobile-carriers-and-oems-get-a.

[25] H. Turner, J. White, J. Reed, J. Galindo, A. Porter, M. Marathe, A. Vullikanti, and A. Gokhale, "Building a Cloud-Based Mobile Application Testbed," Software Testing in the Cloud: Perspectives on an Emerging Discipline, 2012, pp. 382–403.

[26] A. Konstantinidis, C. Costa, G. Larkou, and D. Zeinalipour-Yazti, "Demo: a programming cloud of smartphones," in *Proceedings of the 10th international conference on Mobile systems, applications, and services,* ACM New York, NY, USA, 2012, pp. 465–466.

[27] Windows Phone Dev Center, "Test your app's performance," http://msdn.microsoft.com/library/windowsphone/develop/jj247547(v=vs.105).aspx#BKMK_testperf.

[28] C. Hu, and I. Neamtiu, "Automating GUI testing for Android applications," in *Proceedings of the 6th International Workshop on Automation of Software Test*, ACM New York, NY, USA, 2011, pp. 77–83.

[29] R. Mahmood, N. Esfahani, T. Kacem, N. Mirzaei, S. Malek, and A. Stavrou, "A whitebox approach for automated security testing of Android applications on the cloud," in *Proceedings of Automation of Software Test, 7th International Workshop*, 2012.

[30] T. Haselton, "Android Has 56.1% Of Global OS Market Share, Gartner Says", 2012. Available at: http://www.technobuffalo.com/2012/05/16/android-has-56-1-of-global-os-market-share-gartner-says.

[31] K. Maji, "Characterizing Failures in Mobile OSes: A Case Study with Android and Symbian," in *Software Reliability Engineering (ISSRE) 21st International Symposium*, 2010.

[32] M. Fidelman, "The Latest Infographics: Mobile Business Statistics For 2012," 2012. Available at: http://www.forbes.com/sites/markfidelman/2012/05/02/the-latest-infographics-mobile-business-statistics-for-2012.

[33] D. Franke, C. Elsemann, and S. Kowalewski, "Reverse Engineering and Testing Service Life Cycles of Mobile Platforms," in *23rd International Workshop on Database and Expert Systems Applications (DEXA)*, 2012, pp. 16–20.

[34] D. R. Kuhn, R. N. Kacker, and Y. Lei, Practical Combinatorial Testing, NIST Special Publication, October 2010, pp. 13–15.

[35] Google App Engine, "Developers portal," https://developers.google.com/appengine, accessed Feb. 2013.

[36] "Android SDK," http://developer.android.com/sdk/, accessed Feb. 2013.

[37] "MonkeyRunner tool," http://developer.android.com/tools/help/monkeyrunner_concepts.html, accessed Feb. 2013.

[38] "Localtunnel," http://progrium.com/localtunnel/, accessed 2013.

[39] "VAADIN framework," https://vaadin.com/, accessed Feb. 2013.

[40] "Jersey JAX_RS," http://jersey.java.net/, accessed Feb. 2013.

[41] "Hazelcast," http://www.hazelcast.com/, accessed Feb. 2013.

[42] "AndroidViewClient," https://github.com/dtmilano/AndroidViewClient, accessed Feb. 2013.

[43] "ViewServer," https://github.com/romainguy/ViewServer, accessed Feb 2013.