Toward Automatic Performance Testing for REST-based Web Applications

Chia Hung Kao
Department of Applied Mathematics
National Taitung University
Taitung, Taiwan
Email: chkao@nttu.edu.tw

Chun Cheng Lin
CloudCube Co., Ltd
Taipei, Taiwan
Email: jimlin@cloudcube.com.tw

Hsin Tse Lu
DATA, Institute for Information Industry
Taipei, Taiwan
Email: oliu@iii.org.tw

***Abstract*—Nowadays, more and more web applications are developed to provide their services over the Internet. In addition to functionalities, performance characteristics, including response time, throughput, latency, stability etc., are key factors when selecting appropriate web applications. However, complex architectures, fast changing requirements, strict quality criteria and time to market pressure all impose difficulties and complexities on testing activities. Therefore, for software testers, how to evaluate and ensure performance characteristics systematically and efficiently becomes a critical challenge. In this paper, an approach for automatic performance testing for Representational State Transfer (REST)-based web applications is introduced. Based on Application Programming Interface (API) document and test cases, the proposed approach employs natural language processing (NLP) to parse, match and generate test scripts for performance testing tool automatically. It not only eases the burden of test scripts design, implementation and maintenance efforts on software testers, but also facilitates the execution of performance testing tasks.**

Keywords–Performance testing; web application; software testing.

I. INTRODUCTION

Software companies and service providers develop more and more web applications to provide their services over the Internet. In addition to functionalities, potential users will consider performance characteristics, including response time, throughput, latency, stability, etc. when selecting appropriate web applications for their tasks [1]. Nowadays, in order to fulfill various functional and quality requirements from users, the complexity of web applications is increasing dramatically. Multi-tier considerations, the composition of different software components, architecture concerns, distributed or cluster designs, and data processing mechanisms, all impose design and implementation complexities on web applications. Thus, the difficulties of testing activities arise correspondingly [2]. Furthermore, fast changing requirements and time to market pressure could worsen the situation. In such circumstances, software testers need to frequently create or refine test cases, redesign and implement corresponding test scripts, and execute test scripts to acquire results for further actions. Therefore, how to evaluate and ensure the performance characteristics of web applications systematically and efficiently is a critical issue for software testers [3].

In this paper, an approach for automatic performance testing for REST-based web applications is introduced. It aims to provide software testers with an integrated process from test cases design, automatic test scripts generation, to test execution. Two major software artifacts, including APIs document and test cases, generated from the software development process are used in the proposed approach. APIs document describes information about functionalities provided by specific web application. On the other hand, test cases depict the test scenarios designed by software testers. Through the composition of necessary APIs, the test scenario can be achieved for testing tasks. Based on APIs document and test cases, the proposed approach uses NLP [4] to parse and match corresponding test cases and API, and then generate test scripts for performance testing tool automatically. On the one hand, it eases the burden of test scripts design, implementation and maintenance efforts on software testers. On the other hand, software testers can focus more on the design of test cases and the analysis of test results. Finally, it facilitates the execution of performance testing tasks through automation. Thus, the performance characteristics of web applications can be identified efficiently for further actions on development, operation and maintenance tasks.

The remainder of this paper is organized as follows. Section II reviews related studies. Section III describes the design of the proposed architecture and Section IV presents the usage of the architecture. Finally, Section V presents conclusion and future works.

II. RELATED WORK

In this Section, related studies about automatic test cases generation are introduced. Nébut, Fleurey, Traon, and Jézéquel [5] proposed an approach for automating the generation of test scenarios from use cases. Through the developed transition system, the approach synthesized and generated test cases based on use cases extended with contracts. Jiang and Ding [6] also designed a framework for automatically generating test cases from textual use cases. By using use cases in specific format, the framework built an activity table for constructing EFSM (Extended Finite State Machine), which is the base for test cases generation. Lipka et al. [7] presented a method for semi-automated generation of test scenarios based on textual use cases. The method derived the scenarios from use cases with annotations and then generated the sequence of method invocations for testing. Landhäußer and Genaid [8] proposed the usage of ontology to present the relationship among source code, natural language elements of user stories and test scripts. Through the ontology, the test steps and related artifacts (e.g., APIs and test scripts) can be identified and reused for testing new user stories. Wang et al. [9] proposed an approach that automatically generates executable

 ISBN: 978-1-61208-498-5

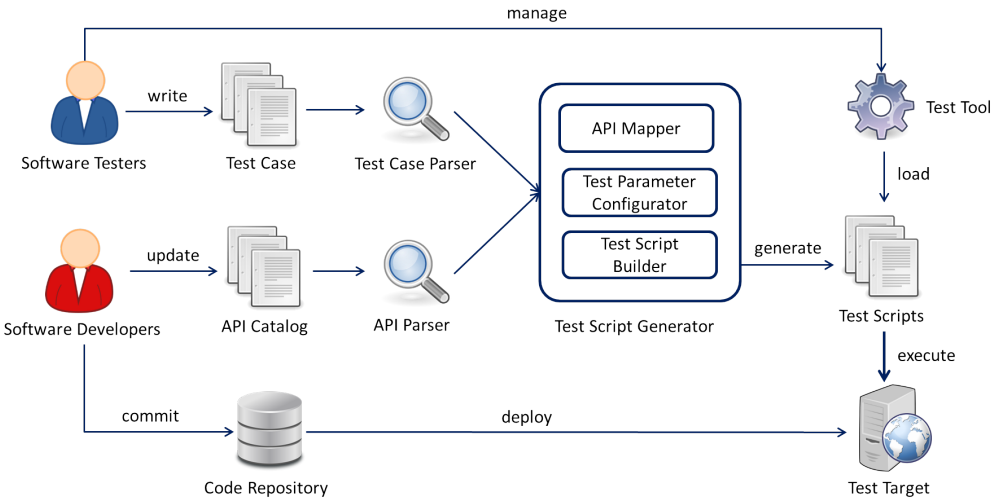

Figure 1. Overview of the automatic performance testing architecture.

system test cases from use case specifications and a domain model. NLP is also used in the approach to extract behavioral information from use cases for test automation. Chen and Miao [10] presented an automatic approach to generate test scripts automatically against JavaScript. The study parsed test cases in XML format and generated Java code for Selenium to execute testing tasks.

To sum up, several studies discussed about the automatic generation of test cases for testing tasks. Based on previous studies, the gap between test scenarios and test scripts is considered in this study through the matching of test cases and APIs document to achieve more automatic testing. In addition, the integrated environment and process for facilitating automatic testing is designed and introduced in this paper.

III. Architecture Design

Fig. 1 depicts the design of the automatic performance testing architecture. Major actors and components are described as follows.

- **Software Testers:** Software testers are responsible for all the testing activities throughout the software development process. Generally, the testing activities include (1) identify test environment, (2) identify performance acceptance criteria, (3) plan and design tests, (4) configure the test environment, (5) implement the test design, (6) execute the test, and (7) analyze results, report and retest [11]. In the proposed approach, software testers can focus on the planning and the design of test cases based on requirements or quality criteria. The implementation and the execution of tests can be achieved automatically.
- **Software Developers:** Based on specific software development process, software developers perform requirement analysis, design, implementation and maintenance tasks to web applications. The implementation will be committed to code repository for version control and continuous delivery. Besides, software developers should update modifications to the API Catalog in the proposed architecture correspondingly.
- **Test Case:** Software testers are responsible for the design, implementation and maintenance of test cases based on requirements and specifications. Generally, test cases include preconditions, test steps, postconditions, and expected results. Test cases can be preserved and managed by test case management systems. In addition, several test case management systems (e.g., Testopia [12]) provide APIs for external access of specific contents within test cases. In the architecture, the test case will be retrieved and analyzed by a test case parser.
- **Test Case Parser:** The test case parser is responsible for analyzing the test steps written in test case. By using NLP, major components can be analyzed and identified, including cardinal number (CD), singular noun (NN), plural noun (NNS), verb (VB), determiner (DT), to (TO), etc. Basically, CD can be considered as the configurations (e.g., number of users and workload) in the performance test case. NN and NNS could be actors, specific objects and the system, respectively. Finally, VB may indicate specific operations of the system. The analysis result can be used to match corresponding APIs and determine configurations in test scripts.
- **API Catalog:** The API catalog helps software developers to create, update, query and maintain API documents for software systems. Famous API catalogs [13] or API management services [14] include Swagger, WSO2 API Management, Apigee, 3Scale, etc. It is anticipated that software developers update API documents to the API catalog once changes happen. Thus, the API information will be kept up to date with the committed code and the deployed test target.
- **API Parser:** The API parser is responsible for analyzing components in an API document. In recent web applications, REST has emerged as the foundation and development principle of the web architecture [15]. A RESTful web service contains three major aspects: the

 ISBN: 978-1-61208-498-5

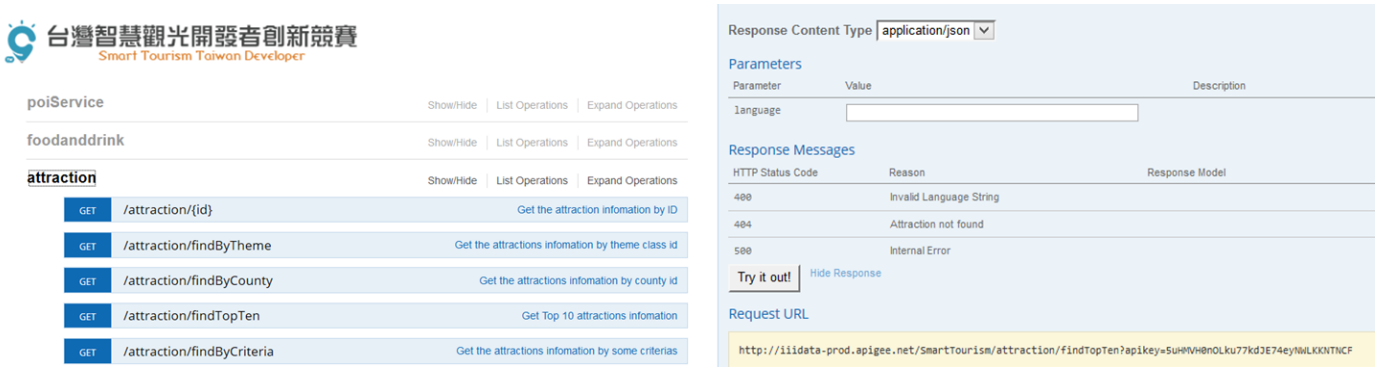

Figure 2. API catalog for web application Smart Tourism Taiwan.

URL of the service, the communication data, and the operations supported by HTTP methods. Through the parsed result, the information about how to invoke the service can be identified.

- **Test Script Generator:** The test script generator is used to generate corresponding test scripts based on test case and API documents. Three major components, the API Mapper, the Test Parameter Configurator and the Test Script Builder, are included in the test script generator. Firstly, the API mapper analyzes the parsed result of test case, searches and maps corresponding APIs based on the information extracted from API catalog. If a specific API is matched, the information described in the API document is parsed and obtained. Secondly, test parameter configurator helps to identify configurations (e.g., number of users and workload) described in test case. Finally, based on the information extracted from test case and API document, the performance test script conformed to specific format of test tool is generated by the test script builder.
- **Test Tool** After the generation of performance test script, the test tool loads and executes the test script to test the target system. In current design, Apache JMeter [16] is selected as the test tool in the performance testing architecture.

IV. Case Demonstration

A web application "Smart Tourism Taiwan" [17] is used to describe the usage of the automatic performance testing architecture. The information of all the APIs are managed by Swagger, and Fig. 2 depicts the screenshot of partial API information. The test case is "1000 users find top ten attractions." Through NLP, "1000" can be identified and used as the input of "ThreadGroup.num_threads" for thread configuration (number of users) in jmx for JMeter. On the other hand, based on the information (API classification, URL and description) from Swagger and NLP result, the API '/attraction/findTopTen' can be identified. Then, the information of "Request URL" can be parsed and used as the input of "HTTPSampler.domain," "HTTPSampler.path," "Argument.name," and "Argument.value" in jmx for JMeter. Based on the content parsed and retrieved from test case and API document, the test script can be built for JMeter for performance testing tasks.

V. Conclusion

Performance characteristics are important factors when users select and use web applications. Due to growing complexity of web applications and the fast changing requirements, efficient and systematic performance evaluation will be the key for further development, operation and maintenance tasks. In this paper, an approach for automatic performance testing for REST-based web applications was introduced. It used NLP to parse and match test cases and API document, and then generate test scripts for the performance testing tool automatically. Through the approach, the burden of software testers can be eased and the performance testing tasks can be facilitated efficiently. A demo case was also introduced to describe the feasibility of the design. Future works include the identification and modeling of test cases to better analyze and realize the purpose of testing tasks. In addition, the API document can be indexed (e.g., by Apache Solr [18]) for better identification, setting and deployment for more flexible and complex test scenarios. Furthermore, the precision of API matching will be analyzed quantitatively and the false positive should be handled. Finally, the overall design will be deployed and evaluated in the development process of various REST-based web applications.

Acknowledgment

This study is supported by the Ministry of Science and Technology of the Republic of China under grant MOST 105-2218-E-143 -001 -.

References

[1] E. J. Weyuker and F. I. Vokolos, "Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study," IEEE Transactions on Software Engineering, vol. 26, no. 12, Dec. 2000, pp. 1147–1156.

[2] A. Bertolino, "Software Testing Research: Achievements, Challenges, Dreams," Proceedings of the 2007 Future of Software Engineering, May 2007, pp. 85–103.

[3] M. Woodside, G. Franks, and D. C. Petriu, "The Future of Software Performance Engineering," Proceedings of the 2007 Future of Software Engineering, May 2007, pp. 171–187.

[4] S. Bird, E. Klein, and E. Loper, Natural Language Processing with Python. O'Reilly Media, 2009.

[5] C. Nébut, F. Fleurey, Y. L. Traon, and J. M. Jézéquel, "Automatic Test Generation: A Use Case Driven Approach," IEEE Transactions on Software Engineering, vol. 32, no. 3, Mar. 2006, pp. 140–155.

[6] M. Jiang and Z. Ding, “Automation of Test Case Generation From Textual Use Cases,” Proceedings of the 4th International Conference on Interaction Sciences, Aug. 2011, pp. 102–107.

[7] R. Lipka, T. Potuák, P. Brada, P. Hnetynka, and J. Vinárek, “A Method for Semi-automated Generation of Test Scenarios based on Use Cases,” Proceedings of the 41st Euromicro Conference on Software Engineering and Advanced Applications, Aug. 2015, pp. 241–244.

[8] M. Landhäußer and A. Genaid, “Connecting User Stories and Code for Test Development,” Proceedings of the 2012 Third International Workshop on Recommendation Systems for Software Engineering, June 2012, pp. 33–37.

[9] C. Wang, F. Pastore, A. Goknil, L. Briand, and Z. Iqbal, “Automatic Generation of System Test Cases from Use Case Specifications,” Proceedings of the 2015 International Symposium on Software Testing and Analysis, July 2015, pp. 385–396.

[10] R. Chen and H. Miao, “A Selenium based Approach to Automatic Test Script Generation for Refactoring JavaScript Code,” Proceedings of the 2013 IEEE/ACIS 12th International Conference on Computer and Information Science, June 2013, pp. 341–346.

[11] J. Meier, C. Farre, P. Bansode, S. Barber, and D. Rea, “Performance Testing Guidance for Web Applications,” Microsoft Corporation, Tech. Rep., Sept. 2007, URL: https://msdn.microsoft.com/en-us/library/bb924375.aspx[accessed: 2016-06-24].

[12] Testopia, URL: https://wiki.mozilla.org/Testopia [accessed: 2016-06-24].

[13] Swagger, URL: http://swagger.io [accessed: 2016-06-24].

[14] A. Acharya, P. Kodeswaran, P. Dey, S. Sharma, and S. Agrawal, “The Talking Cloud: A Cloud Platform for Enabling Communication Mashups,” Proceedings of the 2014 IEEE International Conference on Services Computing, July 2014, pp. 496–503.

[15] R. T. Fielding and R. N. Taylor, “Principled Design of the Modern Web Architecture,” ACM Transactions on Internet Technology, vol. 2, no. 2, May 2002, pp. 115–150.

[16] Apache JMeter, URL: http://jmeter.apache.org [accessed: 2016-06-24].

[17] Smart Tourism Taiwan, URL: http://www.vztaiwan.com [accessed: 2016-06-24].

[18] Apache Solr, URL: http://lucene.apache.org/solr [accessed: 2016-06-24].

 ISBN: 978-1-61208-498-5