TaaS for Improving Quality of Restful Web Services

Shueh-Cheng Hu and I-Ching Chen

***Abstract*—In view of its critical role and popularity in the area of software architecting, Internet of things, and software as a service, a method for improving restful Web service programs' quality obviously is critical and valuable. Consequently, this article presents a service for improving the efficiency of testing restful Web service programs, the corresponding design rationale and impact are described. This kind of services will be helpful in speeding up the testing tasks of restful Web applications, as well as improving quality of restful Web service software, in terms of correctness.**

***Index Terms*—Testing as a service, software as a service, restful web service, software testing.**

I. Introduction

Initially, Web service techniques were invented to loosely couple distributed and heterogeneous software systems, which usually had been implemented with diverse technologies and deployed at different epochs. Basically, there are two kinds of Web service enabling technologies: SOAP (Simple Object Access Protocol) based and REST (REpresentational State Transfer) styled. Due to the overhead of volume message processing, the SOAP-based Web services became bulky to sites with intensive traffic. To address this issue, the REST-styled service presented by Roy Fielding in his dissertation [1] looks a promising approach for architecting various Web services that need to serve vast amount of user requests.

The restful style [2] Web service emerged as an lightweight alternative for realizing the concept of software as a service (SaaS) [3], [4], which is one of the major service delivery models in cloud computing [5] environment. In a SaaS model, functionalities of software are delivered to users through the Internet and HTTP; users do not need to handle executable files and configuration data, instead, these files are hosted in the cloud. Basically, users can consume SaaS via a Web client (browser) that might run on diverse form factors such as smart phones, thin clients, tablet PCs, desktops, and so on. Besides offering a fairer pricing scheme that charge users based on actual usage, the SaaS model accelerated functions and data revision, from end users' perspectives. This feature is very critical to modern e-business and e-commerce platforms where people are always eager for new functionalities for more efficient operations and better competitiveness, but are reluctant to maintain installed software.

Not surprisingly, the aforementioned strengths motivated flagship online enterprises including Google, Yahoo, Facebook, and Twitter that must handle millions of user requests daily, to aggressively adopt the restful approach for developing their Web applications and publish the corresponding APIs with an eye to the performance and pervasiveness. Besides online enterprises, research organizations such as the Lawrence Berkeley National Laboratory also developed a restful API for scientists, thus they can remotely access an array of high performance computing (HPC) resources that were deployed within the laboratory via light-weight mobile devices with Web browser [6]. The international project Globus [7], an grid platform for supporting scientific computing, also applies restful interface for its clients around the globe. An early feasibility prototype using restful Web service in telecommunication industry also indicated that the REST architectural style is also suitable for bridging services across technologies and application domains [8]. Overall speaking, the restful style has being applied to develop Web applications with diverse purposes [9]-[12].

Besides enabling the SaaS service model in cloud computing environments, the restful Web services also play significant roles at the area of Internet of things (IOT) [13]-[15]. IOT refers to various real-world devices (things) could be loosely coupled through embedded computers supporting ubiquitous HTTP protocol; thus, each device could be efficiently tracked and controlled within a coordinated environment to achieve a goal that would not have been achieved without connecting these things. In fact, the IOT concept has been applied in establishment of smart electricity grids [16].

In view of its significance and popularity in the area of software architecting, SaaS, and IOT, a method for improving restful Web service programs' quality evidently becomes critical and valuable. There are volume of research outcome and tools for assuring software quality. In contrast, there are rare research works concentrating on the quality improvement issues of restful Web programs. Accordingly, this article proposes a system, which is presented as a service to users and aims to facilitate the testing of deployed restful Web programs. The anticipated contribution of the present system is to improve the quality of deployed restful Web programs efficiently and effectively.

II. Relevant Techniques and Prior Studies

Before presenting the service-oriented system that automatically tests restful Web programs, the relevant techniques including restful Web services and symbolic execution and prior studies are reviewed.

Manuscript received April 10, 2013; revised June 25, 2013.

Shueh-Cheng Hu is with the Department of Computer Science and Communication Engineering at Providence University, Taichung city, Taiwan 43301, ROC (e-mail:schu@pu.edu.tw).

I-Ching Chen is with the Department of Information Management at ChungChou University of Science and Technology, Yuanlin, Changhua county, Taiwan 51003, ROC (e-mail:jine@dragon.ccut.edu.tw).

A. Characteristics of Restful Web Services

In a restful Web application, everything that could be accessed or operated are treated as resources. The resources must be identifiable via an uniform naming scheme, and the uniform resource identifier (URI) is practically used in all restful Web applications. In contrast with its heavyweight counterpart: the SOAP-based Web service, the restful Web service associates standard HTTP methods with basic operations that intended to be performed on resources. In general, the four basic HTTP methods: PUT, GET, POST, and DELETE have been used to symbolize create, retrieve, update, and delete operations on resources, respectively.

Due to the open and uniform identifying scheme and access (operation) interface, the restful approach significantly reduces the overhead that are caused by the required processing of SOAP-based messages; either message composition or decomposition. Consequently, message recipients can receive and then directly interpret (map) the request sent by users. Such reduction offers users and bystanders a lightweight option for utilizing Web services in the cloud. Moreover, the plain HTTP-based access interface facilitates the integration of popular and lightweight techniques such as asynchronous JavaScript and XML (AJAX) into client side of restful Web applications, which further enables more service consumers using devices with various form factors to access various online services ubiquitously.

In summary, most restful Web applications possess the following characteristics: 1) resource-centered; each component in the system could be treated as a resource that is identifiable via an URI. This features make the Web caching works more efficiently. 2) Uniform access interface; each resource could be created, retrieved, updated, and deleted through the HTTP POST, GET, PUT, and DELETE method, respectively. 3) Representation; each resource could be presented in at least one form, such as XML, HTML, or JavaScript Object Notation (JSON). 4) Connectedness; not only data, the links to other resources also be placed in each resource's representation. 5) Stateless; each operation does not need to rely on prior transmitted data, this feature favors horizontal scalability and cache performance.

B. Automatic Software Testing

To assure the quality of software, verification and/or testing are necessary to be performed before delivering to users. Typically, tasks of software testing include test data preparation, test script execution, and results collection and analysis, all of these are labor-intensive and time-consuming tasks. Consequently, testing consumes considerable portion of resource in most software development projects; the cost of testing exceeds more than forty percent of the total budget according to a report from a leading software vendor [17]. Even with the most allocation of resource, software still cannot be tested thoroughly in most cases due to limited time and budget. In other words, all possible execution paths of software could not be fully covered during testing phase, which definitely will compromise the quality of released software.

To resolve the aforementioned issue, various automatic software testing techniques have been presented to reduce the required manual works and the corresponding cost. Among other automatic testing techniques, the symbolic execution, presented by James C. King in 1976 [18], is a noticeable method for generating test data automatically. Symbolic execution means symbols, rather than real data, are used to traverse a program's execution tree that comprises all execution paths. In a execution tree, each program statement corresponds to a node, each inter-statement transition corresponds to a link between nodes. A path condition comprising input symbols and logical operators is associated with each node and records the condition of reaching the associated node. After excluding all infeasible paths whose path condition is evaluated to false, solving all path conditions associated with feasible paths will obtain a set of test data that can completely cover all of a program’s possible execution paths.

Even with automatic test data generator, there is still a major obstacle in front of us toward an efficient and affordable automatic software testing scheme: state/path explosion [19], [20]. Path explosion represents the increasing number of execution paths will exhaust limited computer memory space eventually, which will halt the system hosting testing jobs. In practical, automatic software testing schemes are resource-intensive, thus, they are only afforded and adopted by large organizations and enterprises [21]. Fortunately, the obstacle has being removed gradually since the inception of testing as a service (TaaS) concept.

C. Testing as a Service

Testing as a service (TaaS) [22], [23], a particular category of SaaS, aims to provide software developers and testers an easy-to-use and cost-efficient testing facility in a form of service. Rationally, TaaS should be deployed in a cloud infrastructure to leverage its advantages, just like other SaaS applications. Consequently, TaaS inheriting merits of SaaS, to certain extent, can solve the difficulties that adopters of automatic software testing techniques are encountering. First, elastic allocation of computing power and storage offered by cloud infrastructures can ease administrative works of an automatic software testing environment; i.e., users only need to focus on their core business: testing the target software. In addition, the pay-per-usage aspect of SaaS service model makes testing jobs much more cost efficient. Second, service-based delivery of testing functionality minimizes the preparation effort before conducting software testing, which further makes testing become a ubiquitous and affordable function. As a result, with TaaS tools, programmers and testers have better chance to identify and correct errors in early stage, which will significantly avoid the higher cost of fixing bugs in late stages of software lifecycle [17].

In view of significance of restful Web programs as well as advantages of TaaS, the present work proposes a system that aims to automatically test the correctness of deployed restful Web programs, and delivers its testing functionality in a form of service. The ultimate goal is providing an environment in which clients can use testing functionality on demand and consume the corresponding computing resources elastically.

III. Testing as a Service for Restful Service Programs

The present system aims to provide service for performing restful Web program testing in an efficient and easy way. This section dissects the system's architecture and describe its design rationales in detail.

A. System Architecture

As Fig. 1 illustrates from a static perspective, there are three major components in the present system: 1) a test data generator that interprets deployed restful Web programs and then generate test data covering the tested programs completely; 2) a test engine that executes scripts in order to send HTTP requests to the deployed programs, along with data being generated if it is necessary to do so. 3) A service wrapper that encapsulate all other components and data modules with a service-based interface (e.g., API), so that clients who might be users or intelligent agents can apply restful styled interface to perform testing tasks.

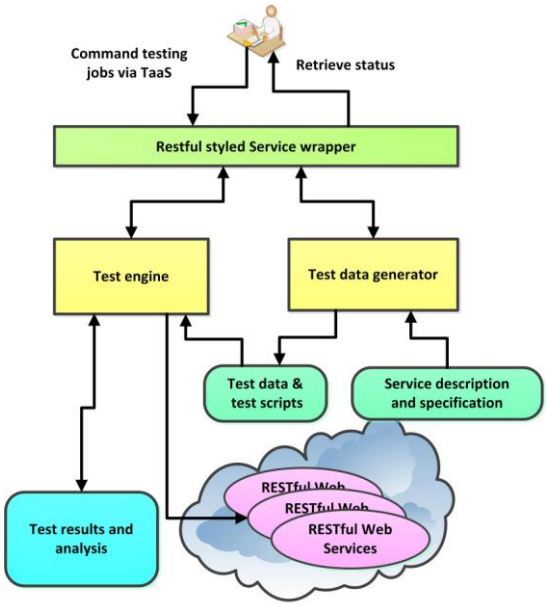

Fig. 1. The TaaS system for restful Web programs.

B. Design Rationales

Apparently, the test data generator needs to rely on symbolic execution technique to generate test data covering deployed restful Web programs completely. However, besides the simple service description including the URI, the detailed specification of service programs, such as data types and ranges, are also necessary to allow the generator come up with both valid and invalid test data sets, which are critical for assuring programs' robustness (capability of exception handling) as well as correctness. Accordingly, the specification of current restful Web programs needs to be extended to take the properties of associated data being processed into account. Recording test data in a machine-readable way is critical for performing test jobs automatically; consequently, an new XML schema needs to be devised for storing generated test data.

In order to execute all feasible paths of tested programs, the test engine needs to read a script file that guiding the engine to perform a sequence of various restful operations and send associated test data, if necessary. Obviously, the test engine must possess the following capabilities in order to achieve the above goal; the first one is applying four HTTP methods (POST, GET, PUT, DELETE) to hit the service programs. The second is processing the XML schemas and documents.

Regarding the service wrapper, it is actually another restful Web application, its main purpose is hiding details of all other functional and data modules with a restful styled interface, so that clients can perform testing tasks via diverse types of devices supporting HTTP-client, which leads to a more accessible testing environment and reduces software defects accordingly.

IV. Impact Analysis

The realization of the proposed system for automatically testing restful Web programs will make impact on multiple facets, which include:

1) The time to market could be shorten; due to the conventionally time-consuming testing phase could be condensed via automatic tools.
2) It is easy to maintain a certain level of testing quality; quality of testing tasks tend to be fluctuating if tasks are performed by humans possessing different levels of expertise, experience, and patience.
3) A more accessible testing service will make software testing more thorough. From temporal perspective, testing works could be conducted within a wider span of lifecycle with TaaS, which means higher chances of bug identification and correction. From spatial perspective, higher coverage of programs' execution paths could be achieved within the same period of time with TaaS since TaaS is more easy-to-use and highly available.
4) Generally speaking, negotiating and then setting a reasonable service level agreement (SLA) before signing service contracts is critical to cloud service providers that need to take both profit and risk factors into consideration. To negotiate and set a reasonable SLA, the cloud service provider, especially SaaS providers, must be able to assess their deployed software's quality precisely and promptly. Obviously, TaaS can provide precise and complete quality data in a more time-efficient way, comparing with traditional testing schemes.

V. Conclusions

Restful Web programming is a light-weight approach toward Internet service provision. Besides being adopted by leading Internet enterprises due to its advantage in terms of efficiency, restful Web applications have being widely applied in the areas of scientific computing, e-commerce, utility management, and many other domains.

Due to its critical roles and popularity, it is important to assure restful Web programs' quality before delivering their functions to users. Considering restful Web programs' fast adoption and increasing installation base, it is irrational to

test them with traditional and labor-intensive schemes.

This article presents a TaaS system that aims to test deployed restful Web programs in a more accessible, efficient, and effective way. To make it more accessible, functionality of the system is wrapped with a restful styled interface. To make it more efficient and effective, the system applied symbolic execution technique to automatically generate test data that cover tested programs completely. Not only fulfilling software testing purposes, the present system could be provided to clients who are interested in particular restful Web service offerings, but need to assess them thoroughly before leasing them.

ACKNOWLEDGMENT

This research work has being funded by the grant from the National Science Council, Taiwan, ROC, under Grant No. NSC 101-2221-E-126 -018. We deeply appreciate their financial support and encouragement.

REFERENCES

[1] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, University of California, Irvine, Irvine, California, 2000.
[2] S. Vinoski, "RESTful Web Services Development Checklist," *Internet Computing, IEEE*, vol. 12, no. 6, 2008, pp. 96-95.
[3] F. Liu, W. Guo, Z. Q. Zhao, and W. Chou, "SaaS Integration for Software Cloud," in *Proc. the 2010 IEEE 3rd International Conference* on *Cloud Computing*, 2010.
[4] G. Goth, "Software-as-a-Service: The Spark That Will Change Software Engineering?" *Distributed Systems Online, IEEE*, vol. 9, no. 7, 2008, pp. 3.
[5] T. Dillon, C. Wu, and E. Chang, "Cloud Computing: Issues and Challenges," in *Proc. the 24th IEEE International Conference on Advanced* Information *Networking and Applications*, 2010.
[6] S. Cholia, D. Skinner, and J. Boverhof, "NEWT: A RESTful service for building High Performance Computing web applications," in *Proc. the Gateway Computing Environments Workshop*, 2010.
[7] B. Allen *et al.*, "Software as a service for data scientists," *Commun. ACM*, vol. 55, no. 2, 2012, pp. 81-88.
[8] C. Fu, F. Belqasmi, and R. Glitho, "RESTful web services for bridging presence service across technologies and domains: an early feasibility *prototype*," *Communications Magazine*, IEEE, vol. 48, no. 12, 2010, pp. 92-100.
[9] P. Belimpasakis and S. Moloney, "A platform for proving family oriented Restful services hosted at home," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 2, 2009, pp. 690-698.
[10] H.-M. Rissanen, T. Mecklin, and M. Opsenica, "Design and Implementation of a RESTful IMS API," in *Proc. the* 6th *International Conference on Wireless and Mobile Communications*, 2010.
[11] S. Gao, H. Yu, Y. Gao, and Y. Sun, "A design of RESTful style digital gazetteer service in cloud computing environment," in *Proc. the 18th International Conference on Geoinformatics*, 2010.
[12] Z. Qian and W. Xianglong, "The research and implementation of a RESTful map mashup service," in *Proc. the 2010 Second International* Conference *on Communication* Systems, *Networks and Applications*, 2010.
[13] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services," *IEEE Transactions on Services Computing*, vol. 3, no. 3, 2010, pp. 223-235.
[14] S. Hodges, S. Taylor, N. Villar, J. Scott, D. Bial, and P. T. Fischer, "*Prototyping Connected* Devices *for the Internet of Things*," *IEEE Computer*, vol. 46, no. 2, 2013, pp. 26-34.
[15] W. Meng, F. Chunxiao, W. Zhigang, L. Shan, and L. Jie, "Implementation of Internet of Things Oriented Data Sharing Platform Based on RESTful Web Service," *City*, 2011.
[16] R. E. Schumann and D. Genoud, "Demand Forecasting and Smart Devices as Building Blocks of Smart Micro Grids," *IEEE, City*, 2012.
[17] S. McConnell, *Code Complete*, Microsoft Press, 2004.
[18] J. C. King, "Symbolic execution and program testing," *Communication of the ACM*, vol. 19, no. 7, 1976, pp. 385-394.
[19] X. Xiao, X.-S. Zhang, and X.-D. Li, "New Approach to Path Explosion Problem of Symbolic Execution," *City*, 2010.
[20] A. S. Douglas, "Simulation-Verification: Biting at the State Explosion Problem," *IEEE Transactions on Software Engineering*, vol. 27, no. 7, 2001, pp. 599-617.
[21] O. Taipale, J. Kasurinen, K. Karhu, and K. Smolander, "Trade-off between Automated and Manual Software Testing," *International Journal of Systems Assurance Engineering and Management*, pp. 1-12, 2011.
[22] G. Candea, S. Bucur, and C. Zamfir, "Automated software testing as a service," in *Proc. the 1st ACM* symposium *on Cloud computing*, Indianapolis, Indiana, USA, 2010, ACM.
[23] L. Ciortea, C. Zamfir, S. Bucur, V. Chipounov, and G. Candea, "Cloud9: a software testing service," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 4, 2010, pp. 5-10.

Shueh-Cheng Hu was born in Taichung city, Taiwan in 1965. He received both B.A. and M.S. degrees in computer engineering from National Chiao-Tung University, HsinChu, Taiwan, in 1987 and 1989, respectively. He received his Ph.D. degree in computer science from Texas A&M University, College Station, TX, in 2000.

Dr. Hu is an associative professor in the Department of Computer Science and Communication Engineering at Providence University, Taichung, Taiwan. Prior to Providence University, he held various software system research and development positions at a number of companies and organizations including AT&T Lab, Taiwan stock exchange corporation, etc. Dr. Hu has been pursuing research in the areas of Web technology, service-based software, e-learning, and e-commerce enabling technologies since 2004. He has published over 30 refereed papers in relevant journals and conferences, which majorly focus on the areas of Software development, Web and cloud technologies, e-learning, and e-commerce.

Dr Hu currently is a member of ACM, IEEE, and IEDRC. He also has served as reviewers for a number of international journals, and committee members of international conferences.

I-Ching Chen was born in Yuanlin County, Taiwan in 1973. She received her B.A. degree in international trade and business from Tunghai University, Taichung city, Taiwan in 1997, and M.S. degree in computer science from the same school in 2002. In 2011, she received her Ph.D. degree in information management from National Yunlin University of Science and Technology, Yuanlin, Taiwan.

Dr. Chen is a faculty member at Chung-Chou University of Science and Technology, where she is an assistant professor in the Department of Information Management. Prior to Chung-Chou University, she held various teaching and administrative positions at Tunghai University and other universities in central Taiwan. She has been pursuing research in the areas of e-commerce, management information system (MIS), customer relationship management (CRM), and e-learning since 2005. She has published over 30 refereed papers in relevant journals and conferences, which majorly explore issues in the areas of MIS, CRM, e-commerce, e-learning, and Web-based software.

Dr Chen currently is a member of IEDRC. She also has served as reviewers for a number of journals, and committee members of international conferences.