A Survey on Security and Performance of SOA via Event Exposure

G.Sasikala[#1],A.Nirmal Kumar[*2]

[#] PG Scholar, Department of Computer Science and Engineering, Christian College of Engineering and Technology, Dindigul, Tamilnadu - 624619, India.
[1]sasiss10@gmail.com

* Assistant Professor, Department of Computer Science and Engineering, Christian College of Engineering and Technology, Dindigul, Tamilnadu - 624619, India.
[2]sa.nirmalkumar@gmail.com

***Abstract*— The Service-oriented applications are services usually collected from various organizations. The details of implementation on services are usually not visible to the service clients, so as to protect the business interests of service providers. This calls for a challenge when compared to white-box test service-oriented applications because it is difficult to determine accurately the test coverage of a service composition as a whole as well as the difficulty to design test cases effectively. This paper focuses on improving the performance of testing and to reduce the delay incurred during testing. To assure the reliability of an SOA application service, security is imposed only on the most critical services of SOA application.**

Index Terms— **Web service composition, white-box testing, event interface, gray-box testing.**

I. INTRODUCTION

The service-oriented architecture (SOA) standard is a widely adopted set of software engineering principles to help manage the complexity of software development for distributed activity applications[14],[15].Service-Oriented Architecture, or SOA, encourages IT departments to switch from application-centric view to a process-centric view. Now, IT departments enjoy the freedom to combine business services from various applications to support for business processes using end-to-end delivery. IT departments can upgrade or change applications without impacting other applications because the integration mechanism of SOA enables loosely coupled integration.

In SOA standard, service providers develop reusable software components, publish them as Web services, and register them in service registries. By composing selected services from registries, service consumers develop composite SOA applications across distributed, various and independent organizations.

To guarantee the quality of SOA applications, integration testing of service compositions is required before the applications are released. Testing is a challenging task, especially, when an SOA application integrates third-party services from different organizations .On the one hand, white box testing of a service composition requires implementation details of every third-party service involved in the composition to be available.

There are two main reasons for this (1) inability to accurately determine test coverage as a whole and (2) the complexity in design test cases effectively. Software engineering is the study that deals with the application of engineering to the design, development, and maintenance of software. Software testing is an investigation carried out to provide information to stakeholders regarding the quality of the product or service being tested. Software testing also provides an independent view of the software which allows the business organizations to manage and understand the risks associated with software implementation.

Testing activities must be carried out throughout the entire SOA project life cycle the activities include design, analysis, planning and execution. Traditional software testing focused only on code-level testing that has evolved from Distributed and Web Service architectures. For testing business logic through the application's user interface Web application testing was introduced, which has proved to be critical when deploying new solutions.

The basic objective of testing is to identify failures of software so that faults may be easily discovered and corrected. The software testing often includes the examination of code and executing that code in different environments and conditions to check for defects or errors. The ultimate goal of Service Oriented Architecture (SOA) is to develop new components and applications .The different roles involved in software testing scenario are manager, test analyst, test designer, tester, automation developer, test lead and test administrator.

The main components of SOA are 1) Service provider 2) Service consumer 3) Service registry. Each component can also act as one of the two other components. For example, if a service provider needs additional information then it can only acquire it from another service, then in that case service provider acts as a service consumer.

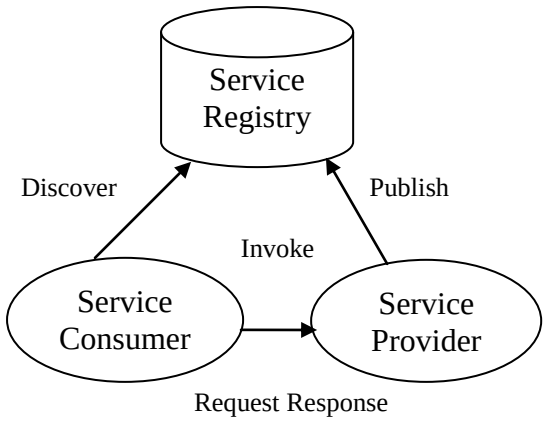

Fig 1: Components of SOA

In this paper ,I proposed a whitening test rather than using white-box testing to improve the performance and security.

As white-box testing focuses on individual functionality which leads to delay and degraded performance whereas whitening test focuses on events rather than functionality. Whitening test provides security only to the most critical events as and when required by the events. Whitening test also has the added advantage of providing recommendations stating which service is the best.

II. TESTING METHODS

Software Testing is the process used for revealing the defects in software. Testing is the group of procedures carried out to evaluate some aspect of a piece of software. Recently there are many methods available in software testing. The methods used for static testing are reviews, walkthroughs, or inspections, whereas dynamic testing refers to executing programmed code with a given set of test cases. Static testing deals with verification, whereas dynamic testing deals with validation. Both types of testing works together to improve the quality of software. The traditional methods of Software testing are divided into two, they are white-box and black-box testing. These two approaches describes how a test engineer designs the test cases.

A. The white-box approach

White-box testing is used to test internal structures or workings of a program. Programming skills as well as a perspective of the system, are used to design test cases in case of white-box testing. To determine the appropriate outputs the tester chooses the correct inputs to exercise paths through the code. A similar concept is used for testing nodes in a circuit, e.g. in-circuit testing (ICT).Testing of the software at the system levels is usually done at the unit level whereas white-box testing can be done only at the unit. White-box testing can be used to test paths within a unit, between units of path during integration phase, and between subsystems during a system–level test. Even though this method of test can be used to uncover many faults or problems, it might not identify missing requirements or unimplemented parts of the specification.

The various techniques used in white-box testing include:1)API testing 2)Code coverage3)Fault injection methods 4)Mutation testing methods5)Static testing methods.

In unit testing, the single component is tested. the main goal is to detect the functional and structural defects in the software unit. In integration testing, several components are tested as a group and it is performed to test the component interaction. In system testing, the system as a whole is tested to evaluate the attributes such as usability, reliability and performance and also to check the specifications or requirements. In user acceptance testing, the software organization must show that the software meets all the users requirements.

B .Block box approach

Black-box testing considers the software as a "black box", monitoring the functionality of the software without any prior knowledge of internal implementation. The testers are not aware of how the software does it or about its operation they are only aware of what the software does. The different methods involved in black-box testing are: equivalence partitioning, boundary value analysis, all-pairs testing, state transition tables, decision table testing, fuzzy testing, model-based testing, use case testing, exploratory testing and specification-based testing.

For testing the functionality of software with respect to the applicable requirements a specification-based testing is used. In case of Specification-based testing it usually requires thorough study of the test cases that are to be provided to the tester. The tester then verifies, the output value (or behaviour), either is or is not the same as the expected value that is specified in the test case that for a given input. Specifications and requirements of the application are the building blocks of test cases, i.e., what it is supposed to do and not how the application does it. The tests can be functional or non-functional, though usually it is considered as functional. Even though specification-based testing is insufficient to protect against complex or high-risk situations, it may be necessary to assure correct functionality, No prior programming knowledge is required which is the main drawback of the black box testing.

B. Grey-box approach

Grey-box testing as opposed to black-box testing involves having prior knowledge of data structures internal to the system and algorithms for the purpose of designing tests. The tester does not have full access to the software's source code. Changing input data and formatting its output do not account for grey-box, since the input and output falls outside the "black box" that is under test. While conducting integration testing between two modules of code written by

two different developers, where only the interfaces are exposed for test that no prior programming knowledge is required this distinction is important. Grey-box testing also includes reverse engineering to determine instances, boundary values or error messages.

Grey-box testing techniques are:1)Matrix Testing 2)Regression testing 3)Pattern Testing 4)Orthogonal array testing.

III. METHODOLOGY

A state of a service is defined as a snapshot of its execution at runtime. The execution of a service can be seen as a series of transitions among its states. The transition from one state to another is defined as a state change. For example, an online shopping service transitions from the state the customer has not been verified to the customer has been verified .

A. *Coverage-equivalent Event Interface*

To make use of event introduction from services to support white-box testing of service compositions. The service providers need to summarize events related to test coverage. It will derive their relationships and declare them in event interfaces.

B. *Coverage Reasoning*

Based on the event interfaces provided by service providers service consumers can monitor the showing events at runtime to determine test coverage. As mentioned in service consumers can construct all the potentially possible observations based on the coverage-equivalent event interfaces from service providers. During testing service consumers generate test cases to test service compositions and pledge to the exposed events from involved services.

IV TECHNIQUES

A. *Load Balancing*

Load balancing can be defined as a networking method used in computers for distributing workloads across multiple computing resources. The resources can be computers, a computer cluster, network links, central processing units or disk drives. Load balancing works to optimize resource use by maximizing throughput, minimizing response time, and avoiding overload of any of the resources. Instead of using single components with load balancing usage of multiple components may increase reliability through redundancy. Load balancing is provided by multilayer switch.

B. *Load Balancer Features*

Hardware and software load balancers may constitute various different features. The main concept of a load balancer is to distribute incoming requests over a number of servers in the cluster. Most of the following features are vendor specific:

1) Asymmetric load: A ratio of load is manually distributed to some backend servers such that some servers have a greater amount of the workload than others.
2) Priority activation: Standby servers can be brought online if the number of servers falls below a certain number, or when load gets too high.
3) TCP offload: Each client accepts HTTP request, which is a different TCP connection. HTTP/1.1 combines HTTP requests from multiple clients into a single TCP socket.
4) TCP buffering: Load balancer gather buffer responses from the server and this data is fed to the slow clients, allowing the web server to release threads for other tasks faster than it.
5) Direct Server Return: It is an optional for asymmetrical load distribution, where request and reply follow different network paths.
6) Content filtering: Balancers arbitrarily modify traffic .
7) HTTP security: Some balancers remove identification headers from HTTP responses, hides HTTP error pages, , and encrypt cookies so that it cannot be manipulated by end users.
8) Client authentication: Before allowing users to access a website the process authenticate users against different authentication sources .

C. *One Time Password*

A one-time password (OTP) is said to be valid for only one login session or transaction. OTPs is used to rectify several disadvantages that are associated with passwords that are used traditionally such as static passwords. The important drawback that is addressed by OTPs is that, they are not vulnerable to replay attacks in contrast to static passwords. A potential intruder who has access to an OTP that is already in use to connect to a service or to conduct a transaction will not be able to modify it, since it will be no longer valid. One shortcoming of OTP is that they are difficult to memorize by human beings. Therefore it requires additional technology to work.

Authentication-as-a-service providers deliver one-time passwords by using various web-based methods. To recognize pre-chosen categories from a grid of pictures generated randomly is a practical example. One registering for the first time on a website, the user chooses categories of things which can be anything from the picture. Each time, when the user logs into the website they are presented with generated grid of alphanumeric characters overlaid on it. The pictures that match with users pre-chosen categories are identified and then the user enters the associated alphanumeric characters to form a one-time access code.

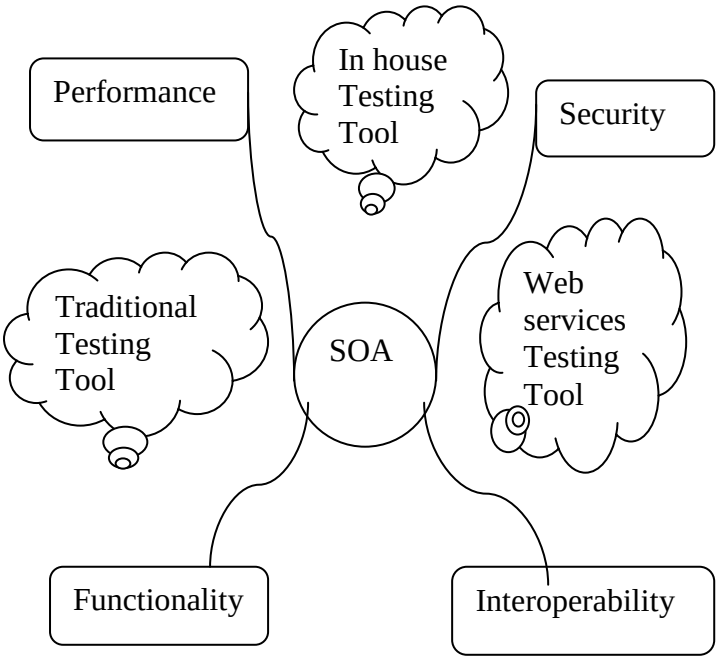

Fig 2. SOA Testing

D. *Security Testing*

Security testing should be carried out throughout the project test phases and not just when the system has been delivered at the end of the project life cycle. To cover all software security issues many organizations perform security penetration testing at the very end of the life cycle. Penetration testing is an authorized technique to detect the security of a system using intruder and/or worm access techniques. When Penetration testing is carried out at the end of the project it leads to a significant amount of risk of not only finding severe security bugs, but also delivering a system that has an inadequate security design.

E. *Software Performance Testing*

To determine how a system or sub-system performs with respect to responsiveness and stability, performance testing is carried out. Performance testing can also be used to investigate, measure, and verify other quality attributes of the system.

V.DISCUSSION

In this paper, we model a service as a finite state machine. In practice, services may be implemented in BPEL and other languages. We can apply many existing work to transform BPEL services into formal models, such as finite state machines and process algebras to just name a few. Some existing Web service standards (like OWL-S) provide such semantics for services (e.g., the pre/post conditions).

A .Asynchronous communication

To ease the presentation and illustration of our approach, we assume services communicate with each other using synchronous communication. Our approach is also applicable to asynchronous communication. To support asynchronous communication, The queues can be introduced in to buffer the asynchronous messages from partners.

B. *Privacy*

In our work, only necessary events are exposed to abstract and reveal coverage-related internal state changes inside a service. All other state changes inside a service and how states are changed (i.e., by what tasks in the business logic) remain invisible to service consumers.

C. Composite Web services

In a service composition, the involved services may be composed of other services. The coverage-equivalent event interface of a composite service should be derived based on the event interfaces of its composed services.

VI RELATED WORK

Survey revealed many related approaches carried out in many areas of study.

The first group proposed a framework for Service Oriented Computing [3]which focuses on the basis of building software by arranging independent and loosely coupled services. Industry has given birth to several standards for specifying and programming such kind of composite services. The problems faced here are there is no much insight about the conceptual kind of testing done and No work discusses strategies for doing integration testing from orchestrations.

The second group focuses on framework for whitening soa testing [2] in which it concentrates on a whitening approach that will make web services more transparent with the addition of an intermediate coverage service. In this paper it performs a preliminary study to show its feasibility and potential value using an instance of SOCT approach .Cost of computation is more in this approach.

The third group focuses on BPEL (Business Process Execution Language)[4] it acts as a de-facto standard. It serves as a standard for web service orchestration that has gained attention from researchers and industries. A semi-formal flow language which deals with complex features such as concurrency and hierarchy is called as BPEL. The operational semantics for BPEL is WSA (Web Service Automata).The drawback associated with this approach is additional test coverage required for checking time and effort of functionality.

The fourth group deals with some ideas for testing the temporal behaviour of real-time systems [8]. The white-box temporal testing which make use of evolutionary techniques to detect system failures in reasonable time and little effort is used in this approach. It results in degraded performance.

The fifth group focuses on a Service-oriented architectures [7]which proposes loosely coupled interacting services as building blocks for distributed applications. Negative test cases are not tested to identify the absence of unintended partners.

The sixth group focuses on a testing approach for SOAs [6] in which a SOA's BPEL business model with pre- and post-condition contracts defining essential component traits, and derive a suite of feasible test cases to be executed after assessing its quality via corresponding coverage criteria. Empirical results for large samples coverage becomes infeasible.

The seventh group first discusses the various state-of-the-art methods for testing SOA applications [9]. The proposed testing architecture consists of several testing units which include test engine, test code generator, test case generator, test executer, and test monitor units. The proposed testing architecture managed to use parallel agents to test heterogeneous web services. Testing non-functional aspects of soa applications are not investigated.

The eight group focuses on Testing Web applications which is recently a challenging work which can greatly benefit from test automation techniques. Ontology is used as a means of test automation. Difficult to maximize the automation of different activities involved in software testing process.

CONCLUSION

White-box testing of service compositions is difficult because service providers usually hide the service implementation details due to business interests or isolation concerns. This paper is based on event exposure from Web services by a unique approach to white-box test service compositions. By deriving coverage-equivalent event interfaces from service implementations, events are defined and exposed from services to accurately determine the test coverage of a service composition at runtime. In this way, service consumers can gain confidence on how adequately a service composition has been tested. This paper also improves the performance and security of testing as well as detects fake websites.

REFERENCES

[1]. Chunyang Ye and Hans-Arno Jacobsen, Senior Member, IEEE"Whitening SOA Testing via Event Exposure" IEEE transactions on software engineering, VOL. 39, NO. 10, OCTOBER 2013.

[2]. C. Bartolini, A. Bertolino, S. Elbaum, and E. Marchetti, "Whitening SOA Testing," Proc. Seventh Joint Meeting of the European Software Eng. Conf. and the ACM SIGSOFT Symp. The Foundations of Software Eng., pp. 161-170, 2009.

[3]. A. Bucchiarone, H. Melgratti, and F. Severoni, "Testing Service Composition," Proc. Eighth Argentine Symp. Software Eng., 2007.

[4]. Y. Zheng, J. Zhou, and P. Krause, "An Automatic Test Case Generation Framework for Web Services," J. Software, vol. 2, no. 3, PP64-77, 2007.

[5]. Peter Dencker,and Rix Groenboom" Methods for Testing Web Services".

[6]. Seema Jehan, Ingo Pill, and Franz Wotawa" Functional SOA Testing Based on Constraints" 978-1-4673-6161-3/13 c 2013 IEEE.

[7]. K. Kaschner and N. Lohmann, "Automatic Test Case Generation for Interacting Services," Proc. Int'l Conf. Service-Oriented Comput-ing Workshops, pp. 66-78, 2008.

[8]. Noura Al Moubayed andAndreas Windisch" Temporal White-Box Testing Using Evolutionary Algorithms".

[9]. Youssef Bassil" Distributed, Cross-Platform, and Regression Testing Architecture for Service-Oriented Architecture" Advances in Computer Science and its Applications (ACSA), ISSN: 2166-2924, Vol. 1, No. 1, March 2012.

[10]. Prachet Bhuyan, Chandra Prakash Kashyap,Durga Prasad Mohapatra" A Survey of Regression Testing in SOA" International Journal of Computer Applications (0975 – 8887) Volume 44– No19, April 2012.

[11]. Antonia BertolinoAndrea Polini" SOA Test Governance: enabling service integration testing across organization andtechnology borders" IEEE International Conference

[12]. Y. Wang, X. Bai, J. Li, R. Huang,"Ontology-Based Test Case Generation forTesting Web Services", ISADS, March2007.

[13]. X. Bai, W. Dong, W. Tsai, and Y. Chen, "WSDL-Based Automatic Test Case Generation for Web Services Testing," Proc. IEEE Int'l Workshop Service-Oriented System Eng., pp. 207-212, 2005

[14]. Ye and H.-A. Jacobsen, "White-Box Testing of Service Compositions via Event Interfaces," technical report, Univ. of Toronto, http://msrg.org/papers/Ye10b. 2010

[15]. G. Canfora and M. Di Penta, "Testing Services and Service-CentricSystems: Challenges and Opportunities," IT Professional, vol. 8,no. 2, pp. 10-17, Mar./Apr. 2006.

[16]. X. Fu, T. Bultan, and J. Su, "Analysis of Interacting Bpel WebServices," Proc. 13th Int'l Conf. World Wide Web, pp. 621-630, 2004.

