Review of the Present State-of-the-Art of Dynamic Test Reconfiguration of Composite Web Services

Sirisha K L S¹, Dr. M. Chandra Mohan², V. Santosh Kumar³

Abstract—In the wake of Service Oriented Architecture (SOA) and its applications that integrated services provided by different businesses, reliability of services play a vital role in the success of those businesses. Testing is the widely accepted approach used to find reliability of services. Testing web services that are composed to form SOA applications is non-trivial and hard to achieve. Web services with BPEL workflow drive most of the businesses that affect all stakeholders. Building comprehensive test strategies for cohesiveness of underlying components and loosely coupled nature of integrated heterogeneous pieces of software is very challenging. Moreover business processes tend to change that makes the job of testing much more complex. Many researchers contributed towards testing of web services. In this paper we review the present state-of-the-art of web service composition methods, testing web services, automatic test case generating and automatic test case reconfiguration. This paper provides insights found in the literature in terms of test methodologies, tools and techniques used for testing web services and web service compositions.

Index Terms—Web services, distributed computing, composite web services, testing, test reconfiguration

1 INTRODUCTION

Service based applications became very popular and they provide certain services to end users. These applications are made up of many similar services (businesses) that seamlessly work together so as to provide quality services to users. Such applications are based on Service Oriented Architecture (SOA) which is distributed in nature. Service oriented applications include applications in banking sector, reservations, e-commerce to mention few. These applications deliver promising services that are essential to end users. Though these applications are very complex, the complexity is transparent to end users. Users enjoy services even without having knowledge on the underlying technologies. SOA based applications need to integrate many related businesses. Supply Chain Management (SCM), ATM network are best examples for that.

Reliability of service oriented applications play vital role to the success of associated businesses. Testing is one of the approaches to find reliability of service oriented applications. The present testing approaches are not adequate to provide the required level of quality assurance in service oriented applications. Stated differently, the testing of service oriented applications needs further research as they are very complex. SOA applications are exposed as atomic web services or composite web services. Atomic web service has a single interface through which other applications can interact.

• Sirisha K L S is currently working in Keshav Memorial Institute of Technology as an Assistant Professor in the Department of Computer Science and Engineering. E-mail: klsirisha@gmail.com
• Dr. M. Chandra Mohan is currently working in JNTUH as a Professor in the Department of Computer Science and Engineering. E-mail: cm_miryalal@jntuh.ac.in
• V.Santosh Kumar currently working in Sreeg Institute of Engineering and Technology as an Associate Professor in the Department of Computer Science and Engineering). E-mail: vennt.santoshkumar@gmail.com

The composite web services on the other hand are made up of multiple web services seamlessly integrated into a business process. Such application may include many services provided by third parties. Therefore testing such applications is non trivial and needs careful selection of suitable testing techniques. There are many reasons for difficulty of testing such applications. First, the services are from different providers and deployed in different servers. Second, they are probably dynamically composed at runtime. Third, the services or the underlying components may be subjected to changes without notice. Fourth, the provider of web service may not have control over the components.

Due to the dynamic nature of the web services, it is essential to have an ongoing process for runtime testing from time to time to ensure reliability. As said earlier it is complex process and needs regular testing as there might be changes to service composition that necessitates the updating of test suits, isolation of composition changes, and notification of testers so as to reconfigure test cases in order to continue testing reliability of those web services. Having understood the full spectrum of the modern service oriented applications, the need for their reliability; in this paper, we throw light on the present state-of-the-art of testing composite web services and the test case reconfiguration to adapt changes. The remainder of the paper is structured into different sections that review the available literature to have useful insights. Our endeavour is to unearth the existing research outcomes and provide research gaps that can help in further research in this area.

2 Changes in Composite Web Services that Affect Test System

While performing reliability testing of composite web services, the test system needs to know whether there are interface changes in the underlying web services and the kind of change that has been made in order to regenerate test cases. King and Ganti [2] investigated different changes that may occur to web
services and categorized them into three types as shown in Table 1.

<table>
<thead>
<tr>
<th>CHANGE CATEGORY</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additive change</td>
<td>It is the change caused due to introduction of new component into an existing SOA application.</td>
</tr>
<tr>
<td>Reductive change</td>
<td>It is the change caused due to removal of a component from SOA application.</td>
</tr>
<tr>
<td>Mutative change</td>
<td>It is the change caused due to transformation of existing component while preserving its functionality to a greater extend. In other words, it is caused due to changes in one of the components.</td>
</tr>
</tbody>
</table>

Table 1 – Different change types in service composition

3. Runtime SOA Testing

Testing SOA applications need live approaches so as to test the reliability from time to time. Bai et al. [4] studied the different runtime environments for SOA applications. They proposed adaptive testing framework that makes use of continuous heuristics to improve testing strategies. Their work is based on the broker architecture explored in [6] and [5]. Their work is an extension to the UDDI test strategy. They used a web service as feedback unit in order to get response while testing and make other consecutive decisions based on the feedback. The Bai et al. improved their work in [7] using distributed test agents in order to control and coordinate activities pertaining to testing. Their work was the first attempt to address issues related dynamic testing of web services. However, their work did not focus on the service compositions. Brenner et al. [8] also investigated into web service testing strategies in similar fashion but focused more on the runtime testing of third party services. They opined that runtime contract testing is one of the strategies to test web services. They listed out different kinds of runtime strategies for testing.

4. Service Composition Testing

Service composition is dynamic and may change in future. This is not visible to the users and web services do not disclose such information. It is impractical to run black box testing on such web services. It is essential to have a look into the internal logic in order to test a functionality of web service. There are tools that came into existing to have service composition testing as explored in [10] and [9]. However, Bucchiaroni et al. [11] opined that these tools and present techniques used for service composition testing are not adequate to handle complex operations that need to address composite data types. Zheng and Yan [41] explored web service composition issues and proposed an algorithm for syntactic matching of web services that are based on planning graph model which eliminates duplicates. Lallali et al. [42] explored on web service composition problems. They proposed automatic timed test case generation for achieving web services composition into a BPEL process. Zheng et al. [43] proposed a recommender system that can provide recommendations to support selection of web services while composing services. Chandra sekaran et al. [44] proposed a tool known as Service Composition and Execution Tool (SCET) composing web services and testing them. They used Web Service Flow Language (WSFL) based specifications to achieve this. Hausmann et al. [45] proposed a model-based discovery of web services formal software models and graph transformations. Tsai et al. [47] explored consumer centric composition of web services. This is collaboration oriented approach in which SOA based web services are identified in consumer-centric fashion. Platzer and Dustdar [49] explored semantic web services and proposed a method for discovering web services. They built a tool known as vector space search engine for this purpose which could browse existing repositories for discovery.

5. Automatic Test Case Generation for Web Service

In case of SOA applications that are made up of web services, it is essential to know interface specification that can be obtained from Web Services Description Language (WSDL). The WSDL based test data generation was explored in [13] and [12]. However, other approaches and specifications in other languages are also available for effective test case generation. Bai et al. [27] explored automatic test case generation approaches based on WSDL. Since WSDL provides useful information on web service and its underlying operations, return types and arguments they focused on WSDL based test case generation. They built Document Object Model (DOM) from WSDL before using it for test case generation. Wang et al. [28] proposed a framework for generating test cases based on ontologies. Their approach was model-driven. They made use of semantic specifications built on OWL-S. They also used Petri nets for achieving model-driven test case generation. Siblini and Mansour [29] proposed a new method known as mutation analysis for testing web services. Their method uses WSDL in order to generate many mutant web service interfaces that are used to generate test cases automatically. Tsai et al. [30] proposed a method that extends WSDL to generate test cases with high coverage. They used four kinds of extensions known as concurrent sequence specifications, hierarchical functional description, invocation sequence, and input-output dependency. Tsai et al. [31] proposed an XML based object oriented framework for automatic test case generation. It converts WSDL specifications into different scenarios from which test cases are captured and generated. However, they approach focused on integration testing and module testing is not given importance. Sneed and Huang [32] proposed a tool for testing web services. The tool was named WSDL Test which is used to generate and validate test data. Thus it can help in accomplishing the task. The tool is finding with simple WSDL files but does not give guaranteed performance with very complex WSDLs. Heckel and Lohmann [33] proposed a method known as contract-based web service testing which exploits matching of service descriptions of provider and requirements of consumer and visualizes contracts. Then the operational interpretation
of rules will help the method to generate test cases. Bai et al. [34] proposed ontology based web services testing method which makes use of a test ontology model which represents test related concepts and their relationships to help in test case generation. The method also has provision for consistency checking and generates less number of test cases. Zhu [35] proposed a framework for testing web services that in SOA applications. In [36] perturbation based testing was proposed. Actually XML messages are modified and thus test cases are explored and generated. This model proved to be efficient but lacks in reconfiguration abilities. Chou and Guo [37] proposed control flow analysis based testing of web services that provide efficient and accurate means for testing besides providing high test coverage. However they do not generate test cases automatically.

Al-Masri and Mahmoud [38] proposed a tool for searching web services over Internet. Web Service Crawler Engine (WSCE) proposed by them generates metadata information pertaining to deployed web services. Morales et al. [39] explored passive testing of web services with timed extended invariants that represent time and data constraints. The method analyzes execution traces for finding formal extended invariants. Brenner et al. [40] performed runtime analysis based testing of web services provided by third parties. They could identify different types of tests that are possible based on the runtime analysis.

Conroy et al. [48] proposed an approach to generate test cases automatically based on the GUI of web service clients. This exploits visualization and accessibility mechanisms to generate unit test cases. Drag and drop and point and click operations were used as basis to identify test case requirements.

6. Testing Web Services

Many researchers contributed to the testing strategies of web services. Zhu et al. [3] proposed a framework for collaborative testing of web services. The collaborative testing web services are provided by third parties. They are discovered and used at runtime using ontology of software testing known as STOWS. Test brokers are used to realize the runtime composition of test services. Ontology is used to represent the services available and the relationship among them. Bultan et al. [14] focused on composite web services and the tracking and analyzing of conversations among such web services. They could identify the differences in synchronous and asynchronous conversations. They could gain knowledge of that for both bottom-up and top-down web service specifications.

Lin et al. [15] threw light on application of safe regression testing on web services built in Java platform. They worked on the white-box testing methods for Java web services using Apache Axis toolkit. Belli and Linschulte [17] proposed an event-based approach for testing web services that contain specific functionality. They made use of sequence graphs and achieve fault management. Tsai et al. [18] focused on group testing of web services using voting algorithm. They used multi-dimensional test data besides clustering them for effective testing of web services. Neisse et al. [19] explored the bandwidth consumption of web services so as to find their feasibility in different conditions. Ciupa et al. [20] focused on a tool named ARTOO for adaptive testing of applications built using Object Oriented (OO) languages. Chen et al. [21] explored adaptive random testing as test selection strategy based on runtime heuristics. Testing non-testable applications [22], testing semantic web services [23], specification based testing of web services that are semantic in nature [24], test case prioritization based on quota [25] and ontology usage for representing composite web services and help in testing [26] are other significant researches on web service testing.

Penetration testing was explored in [46] for securing web services from intrusion attacks. Especially they focused on the testing of SQL injection attacks into web services by using interface monitoring and enhancing attack signatures. They explored SOAP based web services to detect injection vulnerabilities in web services. Yu et al. [50] explored Testing as a Service (TaaS) that runs in cloud for providing testing capabilities to cloud users. This could help could users to have consistent test platform without investing time and money on proprietary testing mechanisms. Kavalli et al. [51] proposed a framework named WebMov for testing composite web services. They employed passive testing techniques to know the robustness and conformance of composite web services. They validated their framework with travel reservation case study.

Location based web services and prioritization of test cases [52], runtime behaviour analysis for conversational web services [53], TGSE tool for testing composite web services [54], a framework for scalable web services [55], investigation of broker role in web service discovery [56], JOpera for testing web services in Agile methodology [57], web services composition based on timed modelling towards automated testing [58], and exploration of web services presence in Internet [59] are other researches focused on web services and the testing of them in SOA environment.

7. Test Reconfiguration for Service Oriented Applications

Once test cases are generated for SOA applications, the reconfiguration of the test cases is essential as there might be different changes in services as explored in [2]. Conroy et al. [1] proposed architecture for test system. They generated test data based on the information available in WSDL content. Their test system architecture includes many components such as test manager, coordinator, WSDL tracker, WSDL resolver, test generator, database coordinator, service client, and change manager. The test manager is responsible for managing entire test process. Coordinator is responsible for coordinating other components in the test system. Change manager is responsible for analyzing changes in the service composition to initiate changes for the test case reconfiguration. Test generator component is used to generate test cases automatically. WSDL tracker is used to ensure that WSDL is available for use. WSDL resolver is used to decompose WSDL and gain knowledge from that. Database controller is used to handle database related queries. Service client is the program built in Apache Axis [16] for testing web services.

Conroy et al. [60] proposed a framework that can be used to reconfigure test cases that have been generated when test compositions are changed. The framework has provision for generating test cases automatically. It makes use of the test composition changes explored in [2] for automatic reconfigu-
ration of test cases. However, the framework supports test cases of operations with simple data types. It does not support complex and composite data types in the web service operations.

8. SUMMARY
This section provides the summary of important research as shown in Table 2. It includes the methods used, their merits and demerits.

| Table 2- Summary of important research on testing web services |
|---|---|---|---|
| Ref. | Method | Advantages | Drawbacks/Limitations | Remarks |
| [3] | Collaborative testing | Reuse of test services | - Uses broker architecture | |
| [14] | Conversation analysis | Runtime behaviour is known | - Differentiation of synchronous and asynchronous calls | |
| [15] | Safe regression testing | Testing internal logic | - Tested only Java web services | |
| [17] | Event-based testing approach | Event sequence graphs help explore | - Fault management | |
| [18] | Group testing | Voting multidimensional data and clustering | - Voting algorithm is used | |
| [27] | WSDL based test case generation | Specification based approach | Dependency on WSDL DOM is used | |
| [28] | Ontology based test case generation | Good representation of knowledge | - OWL-S was used | |
| [29] | Mutation analysis | Test accuracy | Generates many mutated web service | |
| [30] | Extending WSDL | High test coverage | - | Four kinds of extensions are used |
| [31] | XML based OO testing framework | Integration testing | Module testing is not suitable | Converting WSDL specifications into test scenarios |
| [32] | WSDLTest tool | Generates and validates test data | Does not work for complex WSDLs | Uses pre and post conditions |
| [33] | Contract based testing | Useful for unit testing of web services | Works in simulated environment only | Contract rules are automatically interpreted |
| [34] | Ontology based testing | Consistency checking and reducing number of test cases | - | OWL-S is used |
| [36] | Perturbation based testing | Efficient approach | Reconfiguration is not done | Supports document or message-passing style |
| [37] | Control flow analysis | Efficient, accurate and high test coverage | Automatic test case generation is not done | SPARQL queries are used |
| [38] | Web Service Crawler Engine (WSCE) | DisCOVERS web services | Discovery process is not controllable | UDDI business registries are exploited |
| [39] | Passive testing of web services | Invariants representing data and time constraints | Time complexity | Case study based approach |
9. CONCLUSIONS AND FUTURE WORK

In this paper we reviewed literature on the present state-of-the-art of web services testing, SOA applications are composed with proprietary and third party web services. Testing such applications is a challenging task. Different test strategies are required that are specific to web services. The rationale behind the difficulty lies in the fact that web services are in distributed environment and their location is not known prior and discovered at run time. WSDL, SOAP and UDDI are to be understood and WSDL needs to be explored to perform operations and test web services. To reiterate the fact, testing web services is a complex job and that needs systematic approach to ensure reliability of web services. BPEL processes are composed with multiple web services and testing them is hard to achieve. In this paper we reviewed the literature and presented in this paper the insights pertaining to web services composition, testing web services, automatic test case generation, and test case reconfiguration. This research can be extended further to implement automated test case generation for composite web services besides support for automatic test case reconfiguration in adaptive fashion.

REFERENCES


