Slicing of SOA-based Software and its Application to Testing

A Ph.D Synopsis

Submitted to

Gujarat Technological University, Chandkheda

For the Award of Ph.D Degree in

Computer Engineering

Kaushikkumar Keshavlal Rana
Enrollment No: 119997107010

Under supervision of

Prof. Durga Prasad Mohapatra
Department of Computer Science & Engineering, NIT Rourkela.

Co-Supervisor: Prof. Arun K. Somani, Iowa State University, IA 50011.

Abstract

This Ph.D synopsis presents our work concerning slicing of SOA-based software and its application to testing. We first devise a technique for static slicing of SOA-based software based on SoaML service interface diagram. In this technique, we first design a model using SoaML service interface diagram. Then, we create an intermediate representation, called *Service Interface Dependency Graph (SIDG)* from the model that we created. The *SIDG* identifies service call dependency and composite dependency from SoaML model. Giving slicing criterion as an input, our proposed algorithm *SSSIM* traverses the *SIDG* and identifies the affected service interface nodes.

Next, we extend our intermediate representation (SIDG) to be able to compute dynamic slices based on SoaML sequence diagram. In our technique, we first map each message in sequence diagram with the corresponding web service input and output messages. This mapping is static. After that, we construct an intermediate representation of SoaML sequence diagram which we said as *Service-Oriented Software Dependence Graph (SOSDG)*, which is an intermediate representation that needs to be stored and traversed to get a dynamic slice as and when web service gets executed. The SOSDG identifies data, control, intra-service and inter-service dependencies from SoaML sequence diagram and corresponding web service execution. For a given slicing criterion, our proposed algorithm *MBGDS* computes global dynamic slice from *SOSDG* and identifies the affected messages from respective services. The novelty of our work lies in the computation of global dynamic slice based on *SOSDG* and its dependencies induced within and across organizations.

To support the testing of SOA-based software at development phase, we propose an extension to WSDL for carrying out black-box testing. In this context, we apply program slicing artifact i.e dependence graph for testing SOA-based software. In this approach, we impose a hierarchical structure on Web Service Description Language (WSDL) document. The extension to WSDL is carried out by introducing *Web Service Dependence Graph (WSDG)* and dependencies both data and control in XSD (XML Schema Definition) document. This *WSDG* identifies both data and control dependencies from *WSDG*.

Next, we presented two techniques to test SOA-based software using BPMN 2.0 diagram and SoaML service interface diagram. To test it using BPMN diagram, first we convert BPMN diagram into its *Control Flow Graph (CFG)* using our proposed algorithm. Then, we use *Depth First Search (DFS)* method to generate the test paths. Finally, we execute test cases on the generated test paths. The second technique of testing uses SoaML service interface diagram to model SOA-based software. Next, we generate XML Schema and its instance using Visual Paradigm Enterprise tool. At last, we apply test cases based on the schema constraints to test it.

Additionally, we represented a new approach for white-box testing of web services assuming the developer wants to increase the quality of web services and have released service code. In this approach, we create an intermediate representation, called *Service Dependence Graph (SDG)* by merging the *Service Client Dependence Graph (SCDG)* of web service client code and *Web Service Dependence Graph (WSDG)* of web service code. The *SDG* identifies *data dependency*, *control dependency*, and novel dependency that arise at runtime which we named *callee dependency*. Giving slicing criterion as an input, our proposed algorithm, *MBDSWS* computes dynamic slice from *SDG* and identifies the affected statements in both service client and service code. Finally, we execute various test cases using the *MBDSWS*.

Finally, for all the above techniques/approaches we have developed prototype tools to experimentally verify the correctness and preciseness of our proposed algorithms or techniques. The computed slices and generated test cases for several models or web services were found to be correct using both, manual procedure and our prototype tools.

Contents

1 Introduction

Service-Oriented Architecture (SOA) is an architecture for defining how people or machines, business organizations and systems provides and use services to achieve results. Service-Oriented Architecture (SOA) has come out as one of the leading solutions for such business applications.

As more and more business organizations are adopting SOA, and the needs of the business strategies demands rapid delivery and realignment of business services, an organization may overlook the quality aspect of SOA services and may inadvertently introduce errors. It is very important for defect free delivery of SOA based software and their mapping to business requirements. Therefore, an organization must test such applications to ensure high quality SOA solutions and hence ensuring business customers.

Many researchers have given their significant contribution in the field of testing SOA application. In this section we are going to discuss their work. The importance of this topic can be best judge by market prediction reports generated by leading prediction companies. Among them Gartner Incorporation, a leading trend prediction company, was the first one to coin the term ”SOA” and had published in 1996 [25]. It also reports that SOA will be used in more than 50% of new mission-critical operational application and business processes, designed in 2007, and more than 80% by 2010, with success of implementation depending on managing the key challenges faced in testing, debugging. managing and securing, a distributed SOA application.

Another leader, the International Data Corporation (IDC), predicts that the global spent on service oriented software in 2006 was nearly $2 billion , will also rise to $14 billion in 2011[13]. Of course, there are failure reports for example SOA system failure at Heathrow Airports $ 8.6 Billion Terminal 5 Caused 1.6 British Pounds of losses in one week [20], which illustrate the importance of SOA testing.

Gold et al. [22] have briefly analyzed the comprehension scenario in SaaS (Software as a Service) by using a fictional translation service. They also discuss various failures in understanding software, changes and their potential solutions in the context of translation service. They are focused on understanding service failure and how can Alice cope with it rather than finding the errors. Ribarov et al. [18] have discussed the challenges and problem in testing SOA application from micro (object) to macro (service) level. They analyzed testing SOA from the perspective of traditional testing techniques and suggested various solutions. Since SOA is new complex problem tester need novel techniques beyond application of traditional techniques or required new specialized tools.

Hattangadi [10] have discussed SOA testing challenges, tool requirement, change in methodology, and end to end business process testing etc. He proposed 4-step testing approach starting from service, process, end to end and regression testing. His approaches

will never serve as novel information or techniques for SOA testing, it will serve as a handbook or guide. [26] have practically depicted the black, white and gray box testing on SOA web service along with their advantages and disadvantages. In general, the work tells about unit testing of web service lacking of composite and integration testing technique. Morris et al. [8] have analyzed and reports SOA testing challenges , functionality testing, interoperability testing security testing, quality attribute testing and standards conformance testing thoroughly.

Tsai et al. [35] have discussed service testability considering 4 levels as source code, binary code, model and signature. Also they discussed testability of atomic service, data provenance, service integration and service collaboration with a case study. But the services never expose its source code disabling the white box testing. Harris [30] have discussed in detail SOA testing process. He discuss suitability of V-model, revised testing approach and SOA specific testing techniques along with the applicability of traditional testing. Bartolini et al. [5] have proposed SOCT (Service Oriented Coverage Testing) approach consisting service instrumentation, execution, storing log information and retrieval of coverage reports. It consist 3 main components service provider (Testable service), service integrator (SOCT tester) and coverage service provider (Tcov). The SOCT tester calls Testable service, then the SOCT tester receives unique session id (SID). The testable service collects coverage information by calling Tcov. The major flow in this work is the assumption of static binding between SOCT tester and testable service, and service code availability, which may not possible in real SOA.

Inaganti et al. [27] have presented SOA specific testing techniques like service agility testing, BPEL level-1 and level-2 testing, security testing, service design testing and SOA performance testing. They also discuss the SOA road-map process for organizations to become capable of SOA. Dustdar et al. [24] have presented automate testing of SOA through SITT(Service Integration Test Tool), a prototype implementation. They also cover a real world telecom domain example of mobile number portability. It consist 2 phases, in 1st phase, involved web service writes a XML structured log file which is read by TA(Testing Agent) and is sent to MA(Master Agent) via socket interface. In 2nd phase the MA sends back the configuration for TA which is being acknowledged by respective TA. Their work focuses on testing services and workflows by message flow analysis.

Canfora et al. [11] discusses the SOA testability challenges, role of testing and monitoring with image transformation example. In their example they derived 6 equivalence class for SLA constraint (cost 35 $, image= img1.bmp, posterize=true, sharpen=5) and analyzed w.r.t interoperability, run time binding, QoS (Quality of Service) etc. Canfora et al. [12] discuss SOA testing across two dimensions: testing perspectives (service provider, service integrator, third-party certifier, user) and testing level (service functional, service nonfunctional, integration, regression testing).

Teeseling et al. [2] have created a simulation of a business process for a Dutch health law called WMO using WS-BPEL. His work totally focused on simulation SOA with the help of WS-BPEL. Palacios et al. [19] have tabularize their effort on various journal, conferences, workshop, quality assessment questions, data extraction , primary studies, testing objective, distribution of study based on stockholder and in time, technology/standard, validation method , year and source etc. In short, they have exclusively carried out survey in tabular representation.

Linthicum et al. [6] have created and defined SOA test plan, testing domain, architectural objectives, design review and test planning, functional test approach, performance requirements, SLA requirements, data layer testing, service layer testing, policy layer testing, process layer testing, service simulation, core scenarios, user defined compliance rules, SOA testing technology suite, testing execution, looping back to design and development, diagnostics for design-time and runtime, SOA testing debriefs, operational test planning approach etc. These strategies will help practitioner SOA tester to test SOA thoroughly.

Seth et al. [1] have evaluated SOA systems with various size metrics like function point, COCOMO II, and SMAT-AUS framework etc. [29] discusses SOA challenges like visibility, vulnerabilities, quality, lifecycle, impact of changes w.r.t SOA testing. They presented HP quality management solutions like Mercury Virtual User generator, HP load runner, HP QuickTest Professional etc. Davidson [21] have analyzed the Lufthansa cargo case in which Parasoft SOAtest reduces regression testing effort by at 20% than manual testing.[31] reports that Cognizants WSTest Professional, a GUI testing tool for SOA, reduces 40% test script design and 35% were reused test scripts.

2 Motivation

SOA has started gaining importance in this competitive world of business and cloud-based applications. Though, there were predictions about SOA to die out. But, research community along with IT vendors have started substantially contributing to the fast adoption of SOA in the business world.

Between 2005 and 2007, multiple surveys were conducted by organizations such as Forrester, Gartner, and IDC that showed that the top drivers for SOA adoption were mainly internally focused: these top drivers generally included application integration, data integration, and internal process improvement. This is changing. Forrester [9] shows that the number of organizations currently using SOA for external integration is approximately one third of the surveyed organization. While the percentage of externally focused SOA applications is still a minority, this percentage has been growing and the trend will continue as organizations look at SOA adoption for supply-chain integration, access to real-time data, and cost reduction through the use of third-party services via the cloud or *Software-as-a-Service (SaaS)*.

All major IT vendors, a few names, such as IBM, Tibco, Software AG, Sun Oracle, SAP and so on have made huge investments into SOA in recent years, making up a global estimated budget of $2 billion in 2007, which is further expected to rise and reach $9.1 billion by 2014 [13]. Even, corporate giants like Microsoft, IBM, Sun Oracle, SAP, Infosys have already proposed their own SOA solutions and related software products. The current year 2016 is a big boost for SOA as many e-commerce companies adopted SOA for their business processes. This has helped immensely to improve agility and business integration. This also gives an indication that we need more reliable SOA-based systems and their testing is inevitable.

Our observation made us notice that there is a pressing necessity to devise a testing approach for SOA-based software. We also notice that testing approaches have been changed from traditional applications to object-oriented applications and must be changed for SOA-based software too. With this motivation for developing techniques for testing SOA-based software, in the next section, we identify the major objectives of our work.

3 Objectives

The main goal of our research work is to test SOA-based software through program slicing technique during design, testing and maintenance phases of software development lifecycle. To address this broad objective, we identify the following goals:

1. We first wish to compute static slice of SOA software based on SoaML models:
    - Designing suitable intermediate representation.
    - Static slicing algorithm based on proposed intermediate representation.
2. We wish to extend this approach for computing dynamic slices of SOA-based software.
3. Next, we aim to test web services using WSDL.
4. Then, we aim to propose an approach/technique to test SOA-based software using BPMN and SoaML models.
5. At last, we aim to test web services using its code.
6. In addition to investigating our slicing algorithms/techniques/approaches theoretically, we wish to implement all the proposed algorithms/techniques/approaches experimentally for verifying their performance, correctness, and preciseness.
7. Finally, we plan to carry out an analysis of our experimental studies to draw broad conclusions about realized work for time and space requirements.

4 Scope of Our Work

We unanimously decided the scope of our work as to be limited up to Service-Oriented Architecture (SOA) based software testing. It includes which aspects of SOA should be tested (Specification/Model/Code), the elements to be tested (web service, Web Service Description Language (WSDL), XML Schema (XSD), Universal Description Discovery and Integration (UDDI), Simple Object Access Protocol (SOAP)) and other XML artifacts produced during the testing effort. Also, it includes testing strategies for the unit, integration, and systems-level testing of SOA web services.

This means that instead of covering small pieces of testing SOA software development process, we preferred to advance as far as possible on slicing and testing SOA-based software with the aim of being able to present convincing solutions to the SOA testing challenges. Without a doubt, slicing and testing SOA-based software are complex problems as well and deserve to be investigated intensively, which has been done in the scope of our work.

5 Original Contribution by the Thesis

In our work, we have made progress in the field of testing SOA-based software. We have developed novel techniques for solving the previously listed challenges and implemented software prototypes to prove the applicability of our concepts. The most significant contributions and achievements of this thesis are:

- A simple yet powerful technique for slicing SOA-based software, based on models and seems to be intuitive for tester.
- Approaches for testing SOA-based software, that is based on WSDL which provides business flexibility.
- Techniques for testing SOA-based software based on models.
- Techniques to perform regression testing on an SOA-based software.

Moreover, we have published the software prototype as open-source, as a contribution to the research community.

The presentation of novel concepts always requires an evaluation in order to prove their applicability, usefulness, and correctness. Depending on the type of concepts, different types of evaluation make sense to be applied. For this work, however, the evaluation was not trivial. We have performed a comparative evaluation, by matching our approach to other available ones, in order to prove applicability our proposed concepts. This is mainly due to the novelty of our work and the lack of direct competitors. Also, we have not done a precise performance evaluation, as our contribution is not about performance issues nor does

it prove the quality of our approach in any case. It would merely assess the applicability of our prototype implementation, which is primary importance being a proof of concept.

Without a doubt, a real-world evaluation, where our concepts and prototypes are applied in real SOA development projects would make the most sense and give valuable insights into how much the testing process got improved by our contribution. Unfortunately, this was not possible as (i) we did not have access to test a significant number of real-world SOA projects and (ii) it would have been not easy to convince the testers to apply our prototype implementation.

Instead, we evaluated our concepts in a selective manner, choosing what we regarded as reasonable and realizable case studies. For instance, we included a test case generation technique which deals with generating test cases at design time from SoaML service interface diagrams. In contrast to that, we tested SOA-based software at the run-time, which has a significant effect on test results. Moreover, we applied our approach to several standardized case study examples for an internal assessment and as a proof of usability of the prototype implementation. Our contributions and achievements can be judged by next Section.

6 Work Done

The system configuration used to run SSSIM, MBGDS, extended WSDL, BPMN/SoaML Models and MBDSWS algorithms is Windows 7 Professional service pack 1, Intel(R) Core(TM) i3-3240 CPU@ 3.40GHz running at 3.40 GHz, with 4.00 GB RAM. All measured times reported in this section are overall times, including parsing and building of corresponding intermediate representations, building and deploying extended WSDL etc.

6.1 Work-1: Static Slicing of Service-Oriented Software (SOS)

We devise a technique for static slicing of SOA-based software based on SoaML service interface diagram. In this technique, we first design a model using SoaML service interface diagram. Then, we create an intermediate representation, called *Service Interface Dependency Graph (SIDG)* from the model that we created. The *SIDG* identifies service call dependency and composite dependency from SoaML model. Giving slicing criterion as an input, our proposed algorithm *SSSIM* traverses the *SIDG* and identifies the affected service interface nodes.

We have tested the working of SSSIM algorithm using case study examples as stated in [23] with a service call and composite dependencies using SIDG. We studied the run-time requirements of our SSSIM algorithm for these case studies and for several runs. Table 1 summarizes the *average run-time and memory space requirements* of SSSIM algorithm. As we are not aware of the existence of any algorithm for dynamic slicing of service-oriented programs, so we have not presented any comparative results. We have presented only the

results obtained from our experiments. Since we computed the dynamic slices at different service interface nodes of a service interface diagram, we have calculated the *average* run-time requirements of the SSSIM algorithm. The performance results of our implementation agree with the theoretical analysis. From the experimental results, it can be observed that the average run-time increases sublinearly as the no. of service interface nodes increases in a service interface diagram.

Most of the reported work are not directly comparable to our work as they are focused on applying traditional testing techniques, methods, architecture, extension to SOA etc. The only comparable work is of Lallchandani et al. [14, 15]. With respect to asymptotic analysis both perform equally. Lallchandani et al. [14, 15] have merged class diagram and sequence diagram in order to get better information of the system, while we used SoaML service interface diagram. From the implementation perspective, Lallchandani et al. [14, 15] used DOM (Document Object Model) parser, while we used XSLT parser.

6.2 Work-2: Dynamic Slicing of Service-Oriented Software (SOS)

In our technique, we first map each message in sequence diagram with the corresponding web service input and output messages. This mapping is static. After that, we construct an intermediate representation of SoaML sequence diagram which we said as *Service-Oriented Software Dependence Graph (SOSDG)*, which is an intermediate representation that needs to be stored and traversed to get a dynamic slice as and when web service gets executed. The SOSDG identifies data, control, intra-service and inter-service dependencies from SoaML sequence diagram and corresponding web service execution. For a given slicing criterion, our proposed algorithm *MBGDS* computes global dynamic slice from *SOSDG* and identifies the affected messages from respective services. The novelty of our work lies in the computation of global dynamic slice based on *SOSDG* and its dependencies induced within and across organizations.

We have tested the working of MBGDS algorithm using case study examples as stated in [23] with inter-service and intra-service dependencies using SOSDG. We studied the run-time requirements of our MBGDS algorithm for these case studies and for several runs. Table 2 summarizes the *average run-time requirements* of MBGDS algorithm. As we are not aware of existence of any algorithm for dynamic slicing of service-oriented programs, so we have not presented any comparative results. We have presented only the results obtained from our experiments. Since, we computed the dynamic slices at different messages of a services, we have calculated the *average* run-time requirements of the MBGDS algorithm. The performance results of our implementation agree with the theoretical analysis. From the experimental results, it can be observed that the average run-time increases sublinearly as the no. of service increases in a service choreography.

Table 1: Average run-time and memory space requirements of SSSIM algorithm

Sl. No.	Benchmark Model	# Nodes	Slicing Criterion *[SI(n)]*	Avg. Run-Time (in Sec)	Memory Space (in KB)
1	Online Shopping System (OSS) [our case study example]	13	Consumer	105	6.9
2	Hotel Automation Software (HAS)	9	Consumer	100	6.8
3	Medicine Shop Automation Software (MSAS)	9	Provider	101	6.9
4	Bookshop Automation Software (BAS)	8	Consumer	101	6.8
5	Road Repair and Tracking System (RRTS)	8	Provider	100	6.8
6	Restaurant Automation System (RAS)	8	Consumer	102	6.9
7	Student's Auditorium Management Software (SAMS)	8	Provider	100	6.8
8	Library Information System (LIS)	8	Consumer	100	6.8
9	Software Component Cataloguing Software (SCCS)	7	Consumer	100	6.8
10	Supermarket Automation Software (SAS)	7	Consumer	100	6.9
11	Judiciary Information System (JIS)	6	Consumer	70	6.5
12	Municipality Garbage Collection Automation Software (MGCAS)	5	Consumer	50	6.5
13	Motor Parts Shop Software (MPSS)	4	Consumer	40	6.3
14	Railway Reservation Software (RRS)	4	Provider	40	6.3
15	House Rental Software (HRS)	3	Provider	30	6.2

Table 2: Average run-time of MBGDS algorithm

Sl. No.	Name of Case-Study	XMI (#LOT)	# Services	#LOC	Average Run-Time (in Sec)
1	Online Shopping System (OSS) [our case study example]	15708	11	2145	37.45
2	Hotel Automation Software (HAS)	5712	4	795	14.04
3	Bookshop Automation Software (BAS)	9150	6	1189	21.06
4	Road Repair and Tracking System (RRTS)	5812	4	810	13.10
5	Restaurant Automation System (RAS)	5789	5	986	12.22
6	Judiciary Information System (JIS)	5101	4	790	12.11
7	Library Information System (LIS)	9240	6	1190	20.09
8	Software Component Cataloguing Software (SCCS)	5400	3	589	13.87
9	Supermarket Automation Software (SAS)	5119	4	789	14.78
10	Motor Parts Shop Software (MPSS)	5139	3	580	12.45
11	Student's Auditorium Management Software (SAMS)	5333	5	988	13.89
12	Medicine Shop Automation Software (MSAS)	5219	4	794	14.12
13	Railway Reservation Software (RRS)	5318	4	809	14.09
14	Municipality Garbage Collection Automation Software (MGCAS)	5278	5	989	13.18
15	House Rental Software (HRS)	2411	2	397	13.18

To our best of knowledge, no algorithm for *dynamic slicing* of service-oriented software has been proposed so far. We, therefore, compare the performance of our algorithm with the existing algorithms for *static or dynamic slicing* of models/languages. A comparison between the related work is presented in Table 3.

6.3 Work-3: Black-box Testing of Service-Oriented Software (SOS)

We propose an extension to WSDL for carrying out black-box testing. In this context, we apply program slicing artifact i.e dependence graph for testing SOA-based software. In this approach, we impose a hierarchical structure on Web Service Description Language (WSDL) document. The extension to WSDL is carried out by introducing *Web Service Dependence Graph (WSDG)* and dependencies both data and control in XSD (XML Schema Definition) document. This *WSDG* identifies both data and control dependencies from *WSDG*.

We have tested working of our proposed approach using sample WSDLs stated in [37] with structural dependencies incorporated in WSDL. We studied the run-time requirements of our approach for these WSDL data sets and for several runs. Table 4 summarizes the *average run-time requirements* of it. As we are not aware of the existence of any algorithm for WSDL-based testing, we have not presented any comparative results. We have presented only the results obtained from our experiments. Since we black-box tested different WSDLs using their respective consumer programs, we have calculated the *average* run-time requirements of it. The performance results of our implementation are correct and satisfactory. From the experimental results, it can be observed that the average run-time increases sublinearly as the no. of *statement* elements increase in an extended WSDL.

A comparison between the related work is presented in Table 5. We compare the performance of our approach with the existing approaches, techniques or frameworks for WSDL-based testing.

6.4 Work-4: Model-based Testing of Service-Oriented Software (SOS)

we presented two techniques to test SOA-based software using BPMN 2.0 diagram and SoaML service interface diagram. To test it using BPMN diagram, first we convert BPMN diagram into its *Control Flow Graph (CFG)* using our proposed algorithm. Then, we use *Depth First Search (DFS)* method to generate the test paths. Finally, we execute test cases on the generated test paths. The test cases along with its status is shown in Table 6. In the second technique of testing uses SoaML service interface diagram to model SOA-based software. We generate XML Schema and its instance using Visual Paradigm Enterprise tool. At last, we apply test cases based on the schema constraints to test it. Some of the valid and invalid test cases, which we have obtained for our case study OSS, are given below in Table 7.

Table 3: Comparison with related work

Sl. No.	Related Work	Model/ Language	Slicing Entities	Slice Type	Slice Output
1	Zhao [16]	Wright ADL	Software Architecture	Static	Sliced Components
2	Kim et al. [28]	ACME and RAPIDE ADL	Software Architecture	Dynamic	Sliced Components
3	Mohapatra [7]	Object-Oriented Program	Java Prorgam	Dynamic	Sliced objects
4	Panthi et al. [32, 33]	UML	Sequence diagram	Static	Test cases
5	Lallchandani et al. [17]	UML	Class and Sequence diagram	Dynamic	Sliced objects
7	Our MBGDS Algo.	SoaML	Services and Sequence diagram	Dynamic	Sliced messages

Table 4: Average run-time of web service execution

Sl. No.	WSDL Name	XMI (#LOT)	Average Run-Time (in Sec)
1	CGCD (our example)	55	0.24
2	country	901	3.79
3	creditcard	155	0.64
4	currencyconvertor	315	1.50
5	globalweather	238	1.03
6	isbn	154	0.63
7	medicareSupplier	226	0.95
8	periodictable	403	1.66
9	rsstohtml	152	0.61
10	sendsms	177	0.68
11	sendsmsworld	161	0.65
12	statistics	121	0.47
13	sunsetriseservice	123	0.49
14	whois	162	0.62

Table 5: Comparison with related work

Sl. No.	Related Work	Standard/ Technique/ /Approach/ Framework	Dependency	Tool	Output
1	Tsai et al. [34]	object-oriented testing framework	-	-	Test cases
2	Bai et al. [38]	WSDL	input, output and inuput-output	-	Service test case specification
3	Tsai et al. [36]	extended WSDL	input, output and inuput-output	-	Test cases
4	Bartolini et al. [3]	WSDL and XSD	-	soapUI and TAXI	Test cases
5	Bartolini et al. [4]	WS-TAXI framework	-	WS-TAXI	Test cases
6	Our approach	extended WSDL	control and data	Service consumer	SOAP response

Table 6: Test cases execution and its status

Unique Node Number	Element Name	Data Type	XSD Constraints	XSD Constraints Value	Input	Test Case Status
1 & 2	product name	xs:string	minOccurs	1	“ mi3”	Valid
1 & 2	quantity	xs:int	minOccurs	1	“ABC”	Invalid
1 & 2	price	xs:int	minOccurs	1	12000	Valid
1 & 2	seller name	xs:string	minOccurs	1	1	Invalid
1 & 2	contact address	xs:string	minOccurs	1	china	Valid
3 & 4	product name	xs:string	minOccurs	1	1	Invalid
3 & 4	pincode	xs:int	minOccurs	1	384001	Valid
3 & 4	quantity	xs:int	minOccurs	1	1	Valid
5 & 6	shipping company name	xs:string	minOccurs	1	“DHL”	Valid
5 & 6	address	xs:string	minOccurs	1	1	Invalid
5 & 6	charges	xs:int	minOccurs	1	5000	Valid
7 & 8	username	xs:string	minOccurs	1	“ABC”	Valid
7 & 8	password	xs:string	minOccurs	1	“123”	Valid
9 & 10	mobile no	xs:int	minOccurs	1	“ABC”	Invalid
11 & 12	tusername	xs:string	minOccurs	1	“XYZ”	Valid
11 & 12	tpassword	xs:string	minOccurs	1	“test123”	Valid
13	product name	xs:string	minOccurs	1	“123”	Invalid
13	quantity	xs:int	minOccurs	1	1	Valid
13	customer name	xs:string	minOccurs	1	“KKR”	Valid
13	address	xs:string	minOccurs	1	“india”	Valid
13	contact no	xs:int	minOccurs	1	9726677988	Valid
14 & 15	card type	xs:string	minOccurs	1	“CREDIT”	Valid
14 & 15	card no	xs:int	minOccurs	1	“ABC”	Invalid
14 & 15	expiry date	xs:date	minOccurs	1	2020-05-05	Valid
14 & 15	PIN	xs:int	minOccurs	1	“ABC”	Invalid
14 & 15	OTP	xs:int	minOccurs	1	1334	Valid
16 & 17	customer name	xs:string	minOccurs	1	“KKR”	Valid
16 & 17	address	xs:string	minOccurs	1	“india”	Valid
16 & 17	product name	xs:string	minOccurs	1	“mi3”	Valid
16 & 17	contact no	xs:int	minOccurs	1	9726677988	Valid

Table 7: Test case execution on *Online Shopping System (OSS)*

Test Case ID	Element Name	Data Type	Input	XSD Constraints	XSD Constraints Value	Test Case Status
1	seller_name	xs:string	“xiaomi”	minOccurs	0	Invalid
2	seller_name	xs:string	“xiaomi ”	minOccurs	1	Valid
3	seller_name	xs:string	12	minOccurs	1	Invalid
4	product_name	xs:string	“mi”	minOccurs	0	Invalid
5	product_name	xs:string	“mi”	minOccurs	1	Valid
6	product_name	xs:string	12345	minOccurs	1	Invalid
7	quantity	xs:int	12345	minOccurs	0	Invalid
8	quantity	xs:int	12345	minOccurs	1	Valid
9	quantity	xs:int	“ABC”	minOccurs	1	Invalid
10	contact_address	xs:string	“China”	minOccurs	0	Invalid
11	contact_address	xs:string	“ China”	minOccurs	1	Valid
12	contact_address	xs:string	123	minOccurs	1	Invalid
13	price	xs:int	12000	minOccurs	0	Invalid
14	price	xs:int	12000	minOccurs	1	Valid
15	price	xs:int	“XYZ ”	minOccurs	1	Invalid
16	shipping_company_name	xs:string	“DHL”	minOccurs	0	Invalid
17	shipping_company_name	xs:string	“DHL”	minOccurs	1	Valid
19	address	xs:string	“DEF”	minOccurs	1	Valid
20	tusername	xs:string	“test”	minOccurs	1	Valid
21	tpassword	xs:string	“test123”	minOccurs	1	Valid
22	username	xs:string	“test1”	minOccurs	1	Valid
23	password	xs:string	“test123456”	minOccurs	1	Valid
24	pincode	xs:int	4	minOccurs	1	Valid
25	card_type	xs:string	“MAESTRO”	minOccurs	1	Valid
26	card_no	xs:int	123456789	minOccurs	1	Valid
27	expiry_date	xs:date	2016-05-24	minOccurs	1	Valid
28	PIN	xs:int	4554	minOccurs	1	Valid
29	OTP	xs:int	4664	minOccurs	1	Valid

Table 8: Average run-time of MBDSWS algorithm

Sl. No.	Web Service & Service Client (# stmts)	Avg. Run-Time (in Sec)
1	141	0.88
2	120	0.84
3	101	0.76
4	79	0.51
5	62	0.42
6	30	0.32
7	12	0.22

6.5 Work-5: White-box Testing of Service-Oriented Software (SOS)

Table 8 summarizes the *average run-time requirements* of MBDSWS algorithm. Right now the tools accept only a subset of Java constructs. So the program sizes are small. However, the results indicate the overall trend of the performance of our MBDSWS algorithm. Since, we computed the dynamic slices at different statements of both web services and its service client, we have calculated the *average* run-time requirements of the MBDSWS algorithm. From the experimental results, it can be observed that the average run-time increases sublinearly as the no. of statements increases in a web service and its client program.

7 Conclusion

The primary aim of our work was to compute slices and test SOA-based software. We have proposed a novel technique for slicing SOA based on SoaML service interface diagram. Slicing SoaML service interface diagram is difficult due to distributed participant with implicit dependencies among them. We first construct SIDG, an intermediate representation for service interfaces. Our SSSIM algorithm uses SIDG information to compute static slice. Such slice can be used for change impact analysis, testing and understanding distributed architecture such as SOA.

Next we have proposed a novel algorithm for computing dynamic slices of service-oriented programs. We have named our algorithm *Marking Based Global Dynamic Slicing* (MBGDS) algorithm. We consider SoaML sequence diagram and services. Our algorithm uses *Service-Oriented Software Dependency Graph* (SOSDG) as the intermediate representation. The MBGDS algorithm is based on marking and unmarking the edges of the SOSDG as and when the dependencies arise and cease at run-time. Our algorithm does not use any trace file to store the execution history. Also, it does not create additional message nodes

during run-time. This saves the expensive file I/O and node creation steps. Another advantage of our approach is that when a request for a slice is made, it is easily available. We have developed a slicer to verify the proposed algorithm.

Thereafter, we have successfully applied black-box testing strategy to test web service. The novelty of our approach is imposing a hierarchical structure on WSDL, getting dependency information in the form of extensible elements. We have successfully tested our proposed approach.

Then we illustrated the testing approaches for SOA-based software using BPMN and SoaML diagrams. First, we describe our testing algorithm for service-oriented software using BPMN 2.0 diagram, in that we presents CFG generation algorithm and its implementation. Next, we tested test paths of CFG by applying various test cases. Secondly, we present another algorithm for testing service-oriented software using SoaML service interface diagram. In that context, we generated XML schema and its instance. Finally, we tested XML schema instance.

Publication List

Slicing of Service-Oriented Software

1. Web Service Discovery: Concepts, Approaches, Challenges, In *National Conference on Distributed Computing* (NCDC09), 20-21 March 2009, Volume 1, Osmanabad, India, 2009.

2. Dynamic Slicing of Remote Procedure Call (RPC) Program for Debugging, In *Recent Trends in Object-Oriented Software Testing* (RTOOST09), 22-24 June 2009, NIT Rourkela, India, 2009.

3. A Novel Technique For Static Slicing of SoaML Service Interface Diagram, In *The 11th IEEE India Conference* (INDICON 2014), 11-13th December 2014, Pune, India, 2014.

Testing of Service-Oriented Software

4. Black-box Testing of Web Service, In *International Journal of Computer Science & Informatics (IJCSI)*, Volume-3, Issue-2, Page 48-51, 2013.

5. Testing Web Services by Applying Program Slicing, In *International Journal of Advance Research in Computer Science and Management Studies*,Volume 2, Issue 1, January 2014.

Bibliography

[1] Ashish Seth, Himanshu Agrawal, Shim Raj Singla, "Testing and Evaluation of Service Oriented Systems",[online]. Available: *www.arxiv.org*.[accessed on 18/03/2014].

[2] Bart van Teeseling, "Testing Service-oriented Architecture using a simulation", *Master Thesis*, Rijksuniversiteit Groningen, University of Groningen, 2010.

[3] Cesare Bartolini, Antonia Bertolino, Eda Marchetti, Andrea Polini, "Towards Automated WSDL-Based Testing of Web Services", *In the 6th International Conference on Service Oriented Computing, December 1-5, 2008, University of Technology, Sydney, Ultimo City Campus, LNCS 5364, Springer-Verlag Berlin Heidelberg 2008*, Page 524-529, 2008.

[4] Cesare Bartolini, Antonia Bertolino, Eda Marchetti, Andrea Polini, "WS-TAXI: a WSDL-based testing tool for Web Services", *In the International Conference on Software Testing Verification and Validation*, 2009.

[5] Cesere Bartolini, Antonia Bertolini, Sebastian Elbaum, Eda Marchetti, "Whitening SOA Testing", *In the Proceedings of ESEC/FSE 09*, Page 161-170, 2009.

[6] David S. Linthicum, David Bressler, "Key Strategies for SOA Testing", Progress software, [online]. Available: *www.progress.com*. [accessed on 18/03/2014].

[7] Durga Prasad Mohapatra, "Dynamic Slicing of Object-Oriented Programs", *Ph.D Thesis*, Department of Computer Science & Engineering, IIT Kharagpur, May 2005.

[8] Ed Morris, William Anderson, Sriram Bala, David Carney, John Morley, Patrick Place, Soumya Simanta, "Testing in Service-Oriented Environments", *Technical Report*, CMU/SEI-2010-TR-011 ESC-TR-2010-011 March 2010, [online]. Available: *www.sei.cmu.edu* , [accessed on 18/03/2014].

[9] Forrester Research, *Enterprise and SMB Software Survey*, North America and Europe, 2008.

[10] Gaurish Hattangadi, "A Practitioners guide to Modern SOA Testing", *White Paper*, Infosys, July 2011.

[11] Gerardo Canfora, Massimilliano Di Penta, "SOA: Testing and Self-Checking", *International Workshop on Web Services Modeling and Testing (WS-MaTe2006)*, 2006.

[12] Gerardo Canfora, Massimilliano Di Penta, "Testing Services and Service-Centric Systems: Challenges and Opportunities", *IT Pro IEEE*, March-April 2006.

[13] International Data Corporation(IDC), [online], Available: *http://www.idc.com/*, [accessed on 18/03/2014].

[14] Jaiprakash T Lallchandani, R Mall, "Slicing UML Architectural Models", *ACM SIGSOFT Software Engineering Notes*, Volume 33, No. 3, Page 1, May 2008.

[15] Jaiprakash T. Lallchandani, R. Mall, "Static Slicing of UML Architectural Models", *Journal of Object Technology*, Volume 8, No. 1, Page159-188. 2009.

[16] Jianjun Zhao, "Slicing Software Architecture", *Technical Report*, Information Processing Society of Japan, Page 85-92, Nov 1997.

[17] Lallchandani and R. Mall, "A Dynamic Slicing Technique for UML Architectural Models, *IEEE Transaction on Software Engineering*, Volume 37, No. 6, 2011.

[18] Lanchezar Ribarov, Ilina Manova, Sylvia Ilieva, "Testing in a Service-Oriented World", *In the Proceeding of the International Technologies (InfoTech-2007)*, September 21-23, Bulgaria. Volume 1, 2007.

[19] Marcos Palacios, Jose Garcia-Fanjul, Javier Tuya, "Testing in Service Oriented Architectures with Dynamic Binding: A Mapping Study", *Information and Software Technology'*, Elsevier doi:10.1016/j.infsof.2010.11.014.

[20] Mathew Heusser, "Testing Service-Oriented Architecture: A Primer for the Real World", July 31, 2008.

[21] Neil Davidson, "Learn how Lufthansa cargo achieved a problem rate of less than .2% while reducing testing efforts by 20%", *White Paper*, [online]. Available: *www.red-gate.com*.[accessed on 18/03/2014].

[22] Nicolas Gold and Andrew Mohan, Claire Knight, Malcolm Munro, "Understanding Service-Oriented Software", *IEEE Software*, 2004.

[23] Rajib Mall, *Fundamentals of Software Engineering*, PHI Learning Private Limited, second edition, February 2009.

[24] Scharam Dustdar, Stephan Haslinger, "Testing of Service Oriented Architectures- A practical approach", *Springer LNCS*, Page 27-30, September 2004.

[25] "Service-Oriented Architecture Overview and Guide to SOA Research", [online]. Available: *www.gartner.com*, [accessed on 18/03/2014].

[26] "SOA Testing Technique", [online]. Available: *http://www.blog.soatetsting.com*, [accessed on 18/03/2014].

[27] Srikant Inaganti, Sriram Arvamudan, "Testing SOA Application", BPTrends, April 2008, [online]. Available: *www.bptrends.com.*, [accessed on 18/03/2014].

[28] Taeho Kim, Yeong-Tae Song, Lawrence Chung, and Dung T. Huynh, "Dynamic Software Architecture Slicing", *23rd International Computer Software and Applications Conference, COMPSAC '99*, Washington, DC, USA, Page 61-66, 1999.

[29] "Testing Service-Oriented Architecture (SOA) Applications and Services", *White Paper*, [online]. Available: *www.hp.com/go/software*.[accessed on 18/03/2014].

[30] Torry Harris, "SOA Test Methodology",[online]. Available: *http://www.thbs.com/soa*, [accessed on 18/03/2014].

[31] "Testing New-age Application Built on Service Oriented Architecture", *White Paper*. [online]. Available: *www.cognizent.com*.[accessed on 18/03/2014].

[32] Vikash Panthi and Durga Prasad Mohapatra, "Automatic Test Case Generation Using Sequence Diagram", *In the Proceedings of ICAdc, AISC 174*, Springer India, Page 277-284, 2013.

[33] Vikash Panthi, Durga Prasad Mohapatra, "Automatic Test Case Generation Using Sequence Diagram", *International Journal of Applied Information System (IJAIS)*, Volume 2, Page 4, May 2012.

[34] W. T. Tsai, Ray Paul, Weiwei Song, Zhibin Cao, "Coyote: An XML-Based Framework for Web Services Testing", *In the Proceeding of the 7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02)*, 2002.

[35] W.T.Tsai, Jerry Gao, Xiao Wei, Yinong Chen, "Testability of Software in Service-Oriented Architecture", *In the Proceeding of the 30th annual International Computer Software and Applications Conference(COMPSAC06)*, IEEE, 2006.

[36] W.T.Tsai, Ray Paul,Yamin Wang, Chun Fan, Dong Wangl, "Extending WSDL to Facilitate Web Services Testing", *In the Proceeding of the 7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02)*, 2002.

[37] Web Service WSDL data sets, [online]. Available: *http://webservix.net/*, [accessed on 18/03/2014].

[38] Xiaoying Bai, Wenli Dongl, Wei-Tek Tsai, Yinong Chen, "WSDL-Based Automatic Test Case Generation for Web Services Testing", *In the Proceedings of the 2005 IEEE International Workshop on Service-Oriented System Engineering (SOSE05)*, 2005.