CHAPTER - 7

Conclusions

The primary aim of our work was to test SOA-based software through program slicing technique during design, testing and maintenance phases of software development lifecycle. In the following, we summarize the important achievements and contributions of our work. Finally, some suggestions for future work are given.

7.1 Achievements

Our work mainly focuses on proposing appropriate slicing techniques for testing SOA-based softwares during design, testing and maintenance phases of software development lifecycle.

FIGURE 7.1: Use case diagram for our achievements
A use case diagram for our achievements is shown in Fig. 7.1. The use case model comprises two subsystems viz., Technique and Implementation. The Technique subsystem consists of our different static and dynamic slicing algorithms. In the use case model of Fig. 7.1, the use cases have been labelled similar to the names given to each of our proposed algorithms. On the other hand, the Implementation subsystem deals with the prototype implementation of the algorithms proposed in the former subsystem, and its use in carrying out our experimental studies on slicing and testing. The implementation of our proposed algorithms involves development of two prototype tools namely, SOSSS for static slicing of service interface model and SOSDS for dynamic slicing of service choreography model respectively. It also consist open source tools like NetBeans and Altova XML Spy to demonstrate black-box and model-based testing approaches. Each of these use cases have been associated with one of our proposed algorithms in the corresponding Technique subsystem using a stereotype <<implements>>. The use case model depicts an actor which is shown to be a SOA Tester. It shows that an SOA tester can take a SoaML model, SOA artifacts, and SOA technologies to compute slices using our algorithms and respective slicing criterion.

**Achievement of Objectives 1 and 2:**

At design phase, we developed a technique to compute static slices from SoaML service interface diagram. In this technique, we first converted a service interface model into an intermediate representation which we named SIDG (service interface dependency graph) using SoaML service interface diagram. This SIDG identifies service call dependency and composite dependency from SoaML service interface diagram. Giving a slicing criterion as input, our SSSIM (static slicing of service interface model) algorithm traverses the SIDG and identifies the affected service interface nodes.

Further, we devised another technique to compute dynamic slices based on SoaML sequence diagram. In our technique, we first mapped each message in sequence diagram with corresponding web service input and output messages. This mapping is static. After that we constructed an intermediate representation of SoaML sequence diagram which we named SOSDG (service-oriented software dependence graph), which was an intermediate representation needed to be stored and traversed to get dynamic slices as and when web services get executed. This was followed by our slicing algorithm MBGDS (marking based global dynamic slice). The SOSDG identifies data, control, intra and inter service dependencies from SoaML sequence diagram and from web service execution. For a given slicing criterion, the MBGDS algorithm computes global dynamic slice from SOSDG and identifies the affected services. The novelty of our work lies in computation of global dynamic slice based on SOSDG and its dependencies induced within and across organizations and small slices comprising less number of messages.
• **Achievement of Objectives 3 and 4:**

At testing phase, we have developed a new black box testing strategy. In this context we applied program slicing artifacts for testing of service-oriented software based on WSDL document. In this strategy, we imposed hierarchical structure on WSDL (web service description language). The structuring of WSDL is carried out by introducing WSDG (web service dependence graph) and both data and control dependencies in XSD (XML schema definition). This WSDG identifies both data and control dependencies in WSDL. By giving test cases as input to this extended XSD using open source testing tool, a tester will be able to carry out black-box testing.

We have successfully carried out testing of SOA-based software using BPMN and SoaML service interface diagrams. These techniques defines CFG generation, test paths generation, XSD schema generation, XSD schema instance generation, and test cases execution respectively for each models.

We have also developed prototype tools to implement our proposed algorithms and performed experimental studies aligning with Objectives 5 and 6.

7.2 **Contributions**

In this section, we summarize the important contributions of our work. There are three important contributions, Static Slicing of Service-Oriented Software, Dynamic Slicing of Service-Oriented Software, and Black-box Testing of Service-Oriented Software.

7.2.1 **Static Slicing of Service-Oriented Software**

We first developed an intermediate representation for representing simple service-oriented software with higher level abstraction. We have named this intermediate representation Service Interface Dependence Graph (SIDG). We statically constructed the SIDG. Then, we presented static slicing of service interface model (SSSIM) algorithm to compute static slices. Such slice can be used for change impact analysis, testing and understanding distributed architectures such as SOA. We have shown that the space complexity of our algorithms is $O(n^2)$, where $n$ is the number of service interface nodes. The time complexity of SSSIM algorithm is $O(n)$. Further, we have proved that our algorithm computes correct static slices for any given slicing criterion. We have implemented our algorithm to demonstrate it’s correctness experimentally. We have named our slicing tool SOS static slicing (SOSSS).
7.2.2 Dynamic Slicing of Service-Oriented Software

Due to the presence of inter-service and intra-service communications, some control and data flows occurring among the web services, are interdependent. To be able to capture this aspect, we extend our intermediate representation, SIDG to represent the inter-service and intra-service communications. We have named this modified intermediate representation service-oriented software dependence graph (SOSDG). Having introduced SOSDG as an intermediate representation for service-oriented software, we presented a dynamic slicing algorithm for service-oriented software. We named this algorithm marking based global dynamic slicing (MBGDS) algorithm for service-oriented software. The MBGDS algorithm uses SOSDG as the intermediate representation and is based on marking and unmarking the edges of the SOSDG as and when the dependencies arise and cease during run-time. We have shown that the space complexity of MBGDS algorithm is \( O(n^3) \), where \( n \) is the number of statements in the program, and the time complexity is \( O(n^2 m) \), where \( m \) is the length of service choreography execution history (SCEH) involved in service-oriented software choreography (SOSC). We have shown that MBGDS algorithm computes correct dynamic slices with respect to any given slicing criterion. We have developed a slicing tool to verify the correctness and preciseness of our MBGDS algorithm. We have named our slicing tool SOS dynamic slicing (SOSDS). The results obtained from this tool are in line with the presented theoretical analysis results.

An important advantage of our algorithm is that it does not use trace files. Also, it does not create any additional message nodes at run-time. This saves the expensive file I/O operations and message node creation steps. Another advantage of our algorithms is that whenever a request for a slice is made, it is already available in the corresponding data structure. This results in substantial reduction in the response time for slice extraction.

7.2.3 Black-box Testing of Service-Oriented Software

We have successfully applied black-box testing strategy to test web services. Our approach imposed a hierarchical structure on WSDL through web service dependence graph (WSDG) to get dependency information in the form of extensible elements. We have successfully tested our approach using open source tool Netbeans IDE. Our results obtained from this tool Netbeans IDE are in line with the presented technique or approach.

Also, we successfully tested SOA-based software using BPMN and SoaML models. First, we have presented our testing algorithm for service-oriented software using BPMN diagram, in which we presented a CFG generation algorithm and its implementation. Next,
we tested the test paths of CFG by applying various test cases. We have executed the test cases using open source tool *Altova XML Spy*. Secondly, we presented another algorithm for testing service-oriented software using SoaML service interface diagram. In that, we generated XML schema and its instance using *Visual Paradigm Professional Edition* software. Finally, we tested XML schema instance on *Altova XML Spy*. Results obtained for these techniques prove that both these techniques are complementary to each other.

### 7.2.4 Implementation

We have implemented our proposed algorithms and techniques to experimentally verify their correctness. We have tested each slicing tool on a large number of SOA-based software with several executions and slicing criteria. We have observed that our slicers/slicing tools computed correct static or dynamic slices for all input slicing criteria. This experimentally validated the correctness of our proposed algorithms or techniques. We have also implemented our testing algorithms with a number of SOA-based software. We have generated some test cases for different SOA-based software. Finally, we have executed the test cases with the SOA-based software to know whether they are working perfectly or not.

### 7.3 Future Work

Below, we briefly outline the following possible extensions to our work.

- While computing static slices of *service-oriented software* (SOS), we have not considered other SoaML static diagrams like service participant diagram, service contract diagram etc. Our work can be extended by merging theses static diagrams, and creating a unified intermediate representation suitable for slicing.

- While computing dynamic slices, we considered *service-oriented software* (SOS). Our work can be extended to slice *cloud-based software*.

- Our intermediate representation *SOSDG* can be extended to handle composition and exception handling dependencies.

- The *WSDG* extension proposed for *WSDL* can be incorporated in *UDDI* registry.

- The test case generation process using *BPMN* and *SoaML service interface diagram* is manual to some extent, which can be fully automated.
• The slicers can be used to develop efficient debuggers and test drivers for large-scale cloud-based software.

• The proposed algorithms can be used for the development of cloud-based software metrics, and test case generation for cloud-based software etc.
Publication List

Slicing of Service-Oriented Software [Chapter # 3, 4, 5]


Testing of Service-Oriented Software [Chapter # 6]
