An Automatic Machine-Learning Framework for Testing Service-Oriented Architecture

Osama Altalabani

A thesis submitted in partial fulfillment of the requirements of Kingston University for the degree of Doctor of Philosophy

Faculty of Science, Engineering and Computing

Kingston University

April 2014

Abstract

Today, Service Oriented Architecture (SOA) systems such as web services have the advantage of offering defined protocol and standard requirement specifications by means of a formal contract between the service requestor and the service provider, for example, the WSDL (Web Services Description Language) , PBEL (Business Process Execution Language), and BPMN (Business Process Model and Notation). This gives a high degree of flexibility to the design, development, Information Technology (IT) infrastructure implementation, and promise a world where computing resources work transparently and efficiently. Furthermore, the rich interface standards and specifications of SOA web services (collectively referred to as the WS-* Architecture) enable service providers and consumers to solve important problems, as these interfaces enable the development of interoperable computing environments that incorporate end-to-end security, reliability and transaction support, thus, promoting existing IT infrastructure investments.

However, many of the benefits of SOA become challenges for testing approaches and frameworks due to their specific design and implementation characteristics, which cause many testability problems. Thus, a number of testing approaches and frameworks have been proposed in the literature to address various aspects of SOA testability. However, most of these approaches and frameworks are based on intuition and not carried out in a systematic manner that is based on the standards and specifications of SOA. Generally, they lack sophisticated and automated testing, which provide data mining and knowledge discovery in accordance with the system based on SOA requirements, which consequently would provide better testability, deeper intelligence and prudence.

Thus, this thesis proposes an automated and systematic testing framework based on user requirements, both functional and non-functional, with support of machine-learning techniques for intelligent reliability, real-time monitoring, SOA protocols and standard requirements coverage analysis to improve the testability of SOA-based systems. This thesis addresses the development, implementation, and evaluation of the proposed framework, by means of a proof-of-concept prototype for testing SOA systems based on the web services protocol stack specifications. The framework extends to intelligent analysis of SOA web service specifications and the generation of test cases based on static test analysis using machine-learning support.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like thank my PhD advisor, Dr. Souheil Khaddaj, for all his help and support that he has given me over the past years. He allowed me considerable freedom in my research. I feel honoured to have had a chance to work with him and learn from him.

I dedicate this thesis to my family, my wife and my beloved children for their constant support and encouragement.

Chapter 1 – Introduction and Motivation

1.1 Introduction

As SOA technology advances, there is a significant increase in the complexity and sophistication of the major testing procedures, verification and validation of these systems. In industry, these two procedures are often performed manually and not in a systematic manner. They are mainly dependent on the intuition and experience of the tester. However, many open issues remain, in particular systematic validation and testing of SOA components is a widely unexplored field [51]. In general, software testing component observability and controllability are essential factors for component testability. Thus, component requirement understand-ability facilitates test environment knowledge acquisition and helps component test engineers and users to obtain high testability of functional requirements of components so that component test criteria are easily scoped and effective testing can be accomplished [92].

Henceforth, the high testability of SOA components can be achieved by establishing a testability design knowledge which demands requirement knowledge acquisition in order to verify the design of a SOA component, and then to validate its functions, behaviours and performance. This ensures that a component under test meets its functional requirements and specified design in a given operational environment [92].

However, the integration and inter-operability of systems based on SOA have strongly limited the access to efficient testability degree criteria. Thus, the number of possible controllability problems has increased while the restricted monitoring ability has resulted in reduced test observability, making it a problematic for testing. As consequence, testing systems based on SOA presents great challenges to services providers and consumers and

for this reason, testing approaches and frameworks for supporting SOA testability are becoming highly important.

In attempt to solve the practical problems of the lack of SOA testability, knowledge of test design and testing processes is needed. This however, requires knowledge acquisition of system requirements, testing environments during the testing process, and analysis of the results of the testing processes. Hence, this thesis presents a fully automated and systematic approach for testing systems based on SOA supported by Machine Learning (ML) technique and based on knowledge discovery and data mining of protocols and standard requirements and test coverage analysis. Furthermore, the focus of this research specifically includes the web services protocol stack within the testing process, to specifically enable effective automated testing and increase the testability of these systems.

The presented approach builds on a framework of a number of testing modules based on SOA testability knowledge-based (KB) scope. The framework modules analyses the static functional and non-functional (QoS) requirements of the SOA system under test, establishes functional and QoS requirements coverage criteria, and determines whether those criteria have been met beforehand and to what degree. It analyses the testability from using requirements coverage, builds tests criteria based on testability knowledge, acquires information by functional and QoS test simulation and monitoring and acquires test coverage knowledge by comprehensive analysis of the results of the tests. The testability knowledge of test design and processes guided by ML support is important factor presented by the proposed framework, for this reason the reliability of testability

knowledge acquired by the presented framework is high, which has important implications in improving the testability design level for systems based on SOA.

1.2 Challenges

Attentively, any informal system requirement provided by a software specification document is considered the primary source of information for the software engineers for supporting software system integration. With SOA systems, the whole process can be done automatically and systematically. Using WSDL for example, a web service consumer can locate a SOA web services provider and invoke any of its publicly available operations automatically. The WSDL is an XML (Extensible Markup Language) format interface used for describing the web services abstractly, it provides a machine-readable description interface for the functionality offered by a web service, and it describes a web service as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are first described abstractly, and then are bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are then combined into abstract endpoints (services) [2].

Hence, these system interfaces can also be used to derive the test suites for testing purposes. However, to enable an automatic and systematic SOA testing approach, the informal requirements are specified as formal requirements using an appropriate formal specification language, i.e. the WSDL, BPEL, and BPMN [106].

Following a review of related approaches and frameworks in the literature, in this research, the requirements-based functional and non-functional (QoS) testing, the coverage analyses methods of the functional and non-functional requirements and

testing across the middle tiers of SOA systems will be thoroughly investigated and addressed. This can be problematic, since testing the SOA service's front-end user interface becomes irrelevant when it provides no observability of what is actually happening at the SOA service provider or consumer ends. Another problem that will be addressed is associated with the fact that SOA systems are being composed of loosely coupled services at a business-level which are distributed across computer networks, these systems must be tested as end-to-end integration testing methods. Moreover, systems based on SOA require consistent monitoring of testing processes, this include input-out data, and subsequent analysis of test results to determine causes of the defects and recommend solutions. Moreover, such testing approach requires testability knowledge across all heterogeneous environments and across all SOA service providers and consumers, so that a proper fix can applied to the SOA system under test. Furthermore, to accommodate integration errors and defects of the very nature of SOA systems, i.e. service compositions and dynamic service discovery and invocation, which can be validated at run-time only, in this aspect, these systems are required to be monitored at run-time.

These challenges are investigated and addressed in the proposed framework, and implemented by means of a proof-of-concept prototype, this includes empirical analysis of cost-effectiveness and evaluation of the framework. The proposed framework includes a mechanism for incorporating a Multi-Agent System (MAS) with ML systems to support testability of SOA [47], which can play a key role in solving much of the SOA testability problems such as automation, monitoring and quality assurance, and supports proper tests and defect detection. The proposed framework will be widely investigated to verify

and validate SOA-based systems, namely web services implementation. This includes SOA system testing coverage for assessing and finding defects when most cost effective and at all test cycles which include system under test analysis, SOA principle and standard requirements, test requirements coverage, and test coverage measurement.

An empirical study of SOA-based prototypes, at development and testing stages, is used to demonstrate how to apply the proposed testing framework and to work out cost effectiveness measurements for identifying efficient defect detection processes.

Furthermore, to accommodate integration errors and defects of service compositions, and dynamic service discovery and invocation, the research proposes using run-time monitoring approach supported by feedback mechanisms from the ML system, in this way, the interactions among SOA applications can be then translated into test cases that are generated by MAS.

1.3 Research Questions and Objectives

The proposed framework acknowledges that the general concept of the SOA system's testability is limited due to a lack of observation and control of test process and environments. As a result SOA systems still pose many validation and verification problems. These problems are the norms which are the focus of the current research which due to the implementation testing of these systems has become a major problem [106],[47],[78],[8],[68],[91]. The following research questions arise:

1. How to design and develop a testing approach that would increase SOA testability?
2. As consequence of any of approaches for testing SOA systems, a problem that arises is , how to cover functional requirements when validating these systems?

3. How a system based on SOA is validated systematically? What actions have to be taken to ensure a test approach completely and adequately covers all the requirements called for by SOA providers?
4. How can systems based on SOA testing frameworks or tools validate non-functional (QoS) requirements?

In practical steps, the following are the core objectives to meet the aims of the research:

1. Analyse and improve SOA systems testability as practical deliverables for the industry, i.e. not only theoretical.
2. Enable systematic SOA testing and accordingly increase the testability of systems based on SOA standards and principles, rather than refinement of software design and development processes.
3. Use offline-online testing and monitoring methods to meet the lack of the testability factors and improve the testability degree of systems based on SOA.
4. Use automated SOA systems testing in the proposed formwork, as it has been deemed a vital factor in the software industry. Consequently, employing and machine-learning approaches can be a very useful part of these automated testing systems based on SOA.

1.4 Research Contribution

A major contribution of this study is to propose a systematic framework for improving SOA testability. The research demonstrates and supports the hypothesis of using automated software testing by utilising a MAS supported by ML capabilities of reasoning, learning and decision-making. In addition, the work presents several contributions including:

1. An automatic ML framework by means of a proof-of-concept implementation for a testing SOA system, together with a research prototype for proposed frameworks has been developed. This has been developed exclusively for framework implementation and to support the methods in testing, investigation, evaluation, and the outcome of the research work.
2. Improve testing systems as actual deliverables for the industry (top-down integration testing approach) [13].
3. Enable an effective testability degree that is based on combining test case generation with ML and monitoring.
4. Facilitate SOA test automation and increase test productivity.
5. Apply a MAS supported by ML reliability, such as preferences, with purely SOA principle and a standard-based approach using the web services protocol stack. This is essential to increase QoS and level of deployment within both academic and industry sectors.

1.5 Structure of the Thesis

Chapter 2 conducts a thorough investigation into SOA testability problems, leading to the establishment of testability design knowledge and the design of a suitable automated and standard-based framework. The chapter investigates SOA principles, design concepts, and the standards and protocols within SOA, which have emerged to aid and increase the testability of these systems. The chapter concludes with a summary identifying the advantages as well as limitations of the reviewed web services protocol stack. In addition, it identifies the need for a new approach to meet the lack of SOA testability problems and consequently improve the degree of testability.

Chapter 3 investigates and presents what has been done so far in terms of testability including approaches, frameworks, and tools; in order to determine the various approaches and examine how they are applied in various domains for testing systems based on SOA. The chapter also presents an evaluation and concludes with a summary of the approaches.

Chapter 4 presents analyses, design, and implementation of the proposed framework. The chapter proposes a framework to address the challenges that are discussed in Chapter 2. It also presents a suitable approach for improving SOA testability, describes the design and implementation of the proposed framework architecture. The chapter elaborates and describes all the required modules and functional flow processes within a prototype framework. The chapter's findings are summarised in the conclusion section.

Chapter 5 discusses the framework evaluation and the process of evaluation using practical examples including an empirical analysis in order to evaluate the cost-effectiveness of the proposed framework by using key factors such as test cost, defect detection effectiveness and cost-effectiveness measurements. The chapter concludes by discussing the empirical evaluation results and the computational cost-effectiveness of the proposed framework.

Chapter 6 evaluates the proposed framework detailed in Chapter 4 and 5 through an industrial case study. The chapter presents an evaluation of the effectiveness of the proposed framework by practical and systematic implementation on a generic business use case within industry sectors. The industry case study is designed and implemented, as a prototype system that is based on a business use case of an SOA web services environment.

Chapter 7 presents the thesis conclusion by specifying the main contributions of the work in both academia and the industry. It gives a summary of the work done as well as its evaluation results. It also outlines possible research areas that can be carried out in future.

1.6 Publications

During the research, the following papers have been published:

- Altalabani, O and Khaddaj, S. A. (2010), An approach for the testability of SOA and other component-based distributed systems. In: 3rd conference on semantic e-business and enterprise computing; 15 Sept - 17 Sept 2010, Cochin, India.
- Altalabani, O and Khaddaj, S. A. (2011), Automatic Machine Learning Test Case Generation for Service Oriented Architecture, in: 4th conference on semantic e-business and enterprise computing, 2011, London, UK.
- Altalabani, O and Khaddaj, S., (2012), A framework for the testability of service oriented architecture. Journal of Algorithms and Computational Technology, 6(3), pp. 455-472. ISSN (print) 1748-3018.
- O. Altalabani, S. Khaddaj, Test Case Generation for Service Oriented Architecture, in "Enterprise and Cloud Computing: Infrastructure, Applications and Service", pp 43-53, Excel Publications, 2013. ISBN 97881-921320-3-7.
- Altalabani, O and Khaddaj, S. An Automatic Machine-Learning Framework for Testing Service-Oriented Architecture. Manuscript submitted for publication to ACM Transactions on Software Engineering and Methodology journal.

Chapter 2 – Service Oriented Architecture (SOA)

2.1 Introduction

In recent years, SOA have gained significant attention and support from companies in e-business and industry, which are adopting this new design paradigm for increasing IT flexibility and greater reuse of existing assets. More specifically, SOA defines sets of rules and capabilities as design principles, standards, and protocols that must be obeyed in order to take advantage of the services [42],[44],[119]. These rules define how to integrate widely disparate applications that are interconnected and integrated within wide multiple networks and use multiple computers.

In order to understand and to improve the testability of SOA, which lead to design a suitable automated and standard-based framework, this chapter conducts a thorough investigation into SOA design concept and principles, standards, and protocols in attempt to establish a testability design knowledge and to identify SOA testability problems and consequently improve the testability degree. The chapter concludes with a summary identifying the advantages as well as limitations of the reviewed web services protocol stack. In addition, it identifies the need for a new approach to meet the lack of SOA testability problems and consequently improve the testability degree.

2.2 SOA Historical Background

The principal concept of SOA dates back to the 1960s. The Component Object Model (COM), Distributed Component Model (DCOM), Common Object Request Broker Architecture (CORBA), Remote Method Invocation (RMI), and Electronic Data Interchange

(EDI) are familiar examples of component-based distributed architectures [80]. However, these older examples of distributed computing platforms are subject to a number of problems. First, they are tightly coupled, which meant that both ends of each distributed system interface had to agree on the details of the updates or upgrades of the Application Programming Interfaces (APIs). Secondly, these Service-Oriented Architecture software are proprietary; while DCOM controlled by Microsoft, CORBA supposedly is an open standards effort, but in practice implementing CORBA architecture added the restriction of working only under a single vendor's implementation and specification [26]. Nowadays, SOA represents a new generation of distributed computing platforms. SOA builds upon previous distributed system platforms, adding new design principle layers, governance roles, and a wide set of standards and protocols. In addition to that, defining and publishing a public technical interface in terms of a service contract, which is considered the most fundamental part of service-orientation. A service contract is comprised of one or more of service technical description documents provided by a SOA system to access services protocols, functionalities, and end points entries for SOA implementation [86].

At present, web services are a key breakthrough to support the openness, heterogeneity, and flexibility of SOA systems, but there is still a big gap between the underpinnings of the architectural style and its supporting technology. The architectural style embodies dynamism and flexibility, while supporting technologies are still static.

Web services can be implemented and integrated widely with disparate applications on the web in heterogeneous platforms [32]. SOA does not limit consumers to any particular transport or medium in order to consume services—it could be Hypertext Transfer

Protocol (HTTP) through Internet, Java Message Service (JMS), File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), or any protocol. SOA components can be deployed on different types of hardware and platforms (see the Figure 2.1). On the other hand, these components have to use and agree upon a known protocol. SOA components can be developed using such technology disregarding the idea that other components have to understand or even know about other components' technologies [86].

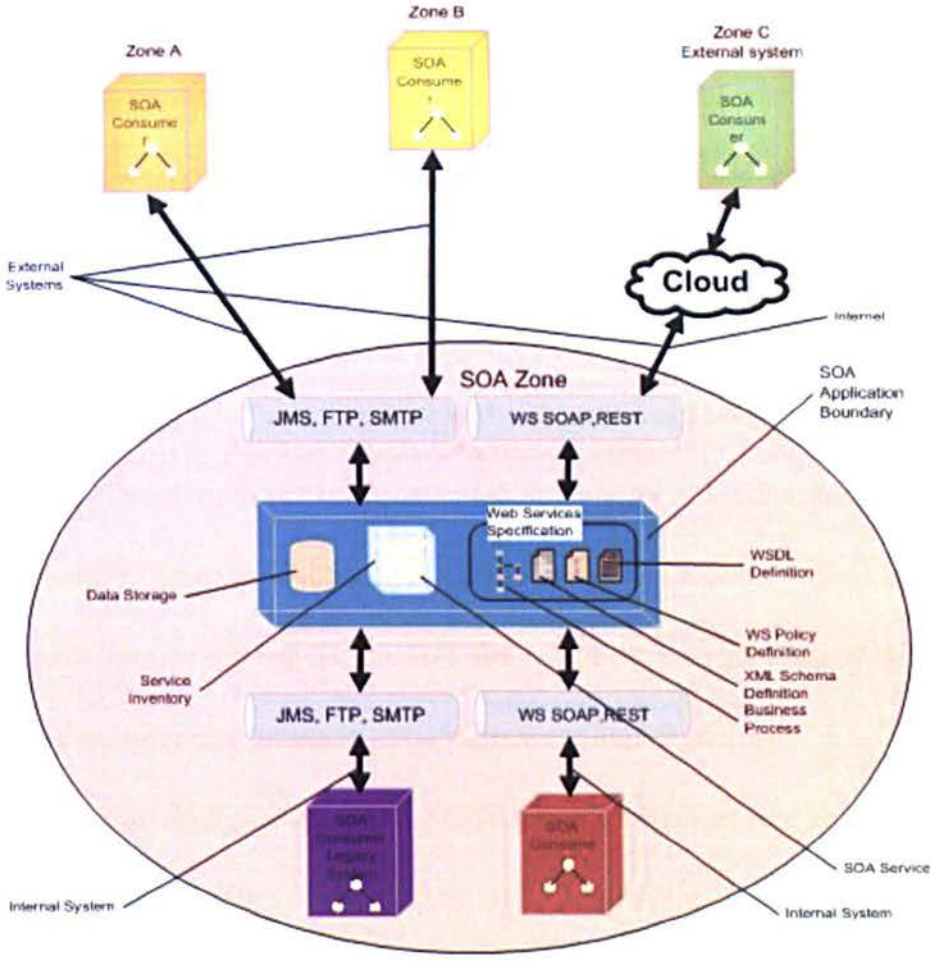

Figure 2.1. SOA implementation

2.3 SOA Design Principles

In order to design and implement SOA solutions, SOA services generally need to adhere to the following principles [86],[32]:

1. Service abstraction (advocates for exposing minimal amount of information about a service from the outside world).
2. Service autonomy (advocates for controlling the shared access of service resources and service logic encapsulation).
3. Service composability (concerns about the ability of any service to act as an effective composition participator, regardless the size and complexity of the composition procedure).
4. Service discoverability (advocates for effective and manageable service discovery by different kind of users, with or without technical background).
5. Service formal contract (advocates for maximising the adherence of Service Level Agreement (SLA) for delivering SOA services within a given service inventory).
6. Service loose coupling (advocates for minimising service coupling requirements and dependencies between service provider and consumer).
7. Service reusability (promotes for full support of services reusability).
8. Service statelessness (advocates for minimising the amount of resource consumption due to service states management in order to remain available to other concurrent consumers).

2.4 SOA Design Concept

In contrast to conventional software architectures which primarily define the software components of a system in its subsystems and their interconnections, SOA systems capture the software design concept as external visible components in a logical way. External visible components support the concept of software components can make of a system architecture that will work together to implement the overall system, such as providing services to either end-user applications or other services distributed in a network through published and discoverable interfaces [58].

SOA supports designing and implementing software in the form of interoperable services which support an IT business architecture model called service-orientation, where service-orientation is about solution logic which provides software capabilities as discoverable and composable services through interfaces called service contracts [112], [93].

Service-orientation provides a set of design principles to ensure the manner in which it carries out the separation of operations of the SOA system components, in order to handle the complexity and to achieve the required software quality factors [93]. SOA is governed by these principles. Applying service-orientation design principles results in the components of SOA system being partitioned into operational capabilities, each designed to solve an individual concern, such as run-time interoperability, loose-coupling, re-usability of SOA services when implementing new businesses or extending life-spans of existing systems. The overall goal of SOA is to support the idea of run-time integration and loosely coupled services across heterogeneous platforms and throughout distributed

environments. Consequently, SOA systems have the advantage of improving the flexibility of system design, development and implementation.

SOA promotes the use of protocols and standards, which are critical in any integration because protocols and standards create a common baseline between SOA service providers and consumers to work on. In addition, the compliance provided by SOA enhances the integration experience with the flexibility to compose, change, or update services almost seamlessly to clients with SOA's decoupling capabilities [93].

2.4.1 Generic Software Architecture

In software engineering, conventional software architecture is the process of analysing, identifying, and presenting a structured software system that can be successfully designed and implemented according to the requirements analysis of the software components. Each system is composed of subsystems, which in turn are made up of other subsystems, each subsystem being delineated by its boundaries. The interconnections and interactions between the subsystems are termed "Interfaces". Interfaces occur at the boundary and take the form of inputs and outputs. The process also includes optimisation of the quality of software system's characteristics in general, such as performance, security, reliability, and so on [58].

Software architecture is determined based on a wide range of principles and each of these principles can have considerable standards and specifications, which can affect software architecture design and implementation, and overall quality of the software [63].

2.4.1.1 Generic SOA Design Layers

In a typical conventional SOA architectural scenario, a web service system employs basic service capabilities provided in the basic service layer, e.g. building a simple weather report web service. It defines an interaction between A service broker (service discovery agency) as an exchange of messages between service consumers (clients) and service providers. These interactions involve the publishing, finding and binding of operations [50]. A service provider hosts the web services within a computer network as an accessible software component and as an implementation of the given web services [93].The web services provider defines the description of the service and publishes it to a requestor or to the service broker through which the web service description is published and made discoverable.

The web services then require two fundamental operations: find and bind. The service requesters find the required service using a service broker and bind to it. The service client retrieves the service description typically from the registry or repository (instance of a service broker) such as Universal Description, Discovery and Integration (UDDI), and uses the service description the WSDL to bind with the service provider and invoke the service or interact with service implementation [72].The conventional SOA web services architectural design idea is shown in Figure 2.2.

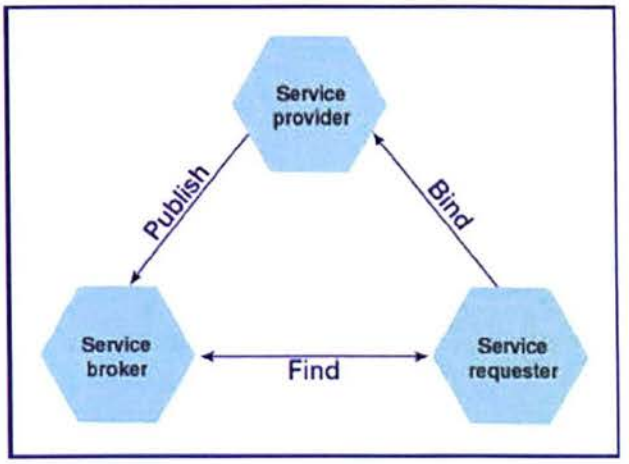

Figure 2.2. Generic SOA design layer

In order to perform the three operations of publish, find and bind in an interoperable way, the web services must adapt web service stack standards and protocols of the service layer at each level of web service operations [44]. The foundation layer of the web services protocol stack's service layer is the web service's network which must be accessible to be invoked by a service requestor. Web services that are publicly available on the Internet use commonly deployed network protocols. Due to its predominance, HTTP is the de facto standard network protocol for web services which are using the Internet as a network interconnection. Similarly, other Internet protocols can be supported, including SMTP and FTP. The next web services protocol stack's service layer is the XML-messaging layer which uses the XML as the basis for the messaging protocol called Simple Object Access Protocol (SOAP). SOAP is a simple standardised HTTP POST with an XML payload envelope mechanism for communicating document-centric messages and remote procedure calls. SOAP incorporates defined extensions to the message envelope using SOAP headers and a standard encoding of operation or function. SOAP messages supports publish, find, and bind operations in the web services architecture.

The next layer is the service description layer which is actually a stack of description documents called the WSDL. The WSDL is the de facto standard for XML-based service description. It can define the interface and mechanics of service interaction as a minimum standard service description which is necessary to support interoperable web services. Additional descriptions can be specified for business context, qualities of service and service-to-service relationships.

The WSDL document can be complemented by other service description documents to describe these higher level aspects of the web services. For example, a business context is described using UDDI data structures in addition to the WSDL document. Service composition and flow are described in a PBEL and BPMN document.

Due to the fact that a web service is defined as being network-accessible via SOAP and represented by a service description, the first three layers of the stack are required to provide or use any web services. The simplest web services stack would consist of HTTP for the network layer, the SOAP protocol for the XML messaging layer and WSDL for the service description layer. This is the generic interoperable SOA web services design layer that all inter-enterprise or public web services should support. Web services especially intra-enterprise, or private, web services, can support other network protocols and distributed computing technologies. Figure 2.3 depicts the generic interoperable SOA web service's design layers.

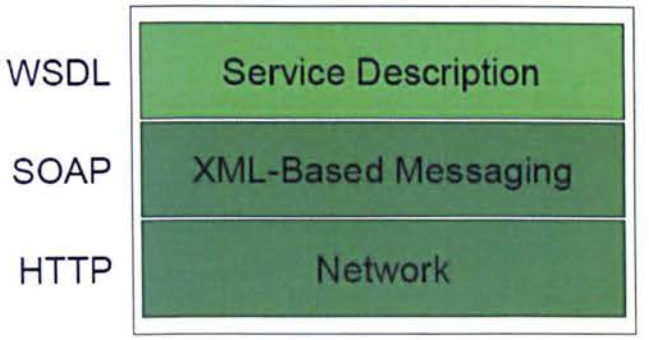

Figure 2.3. Generic interoperable SOA web services design of the service layers

2.4.2 Web Services and their Categories

In many cases, SOA systems are typically built from web services. However, any service-based technology may be used. Web services are currently the preferred standard base model to represent SOA implementation [44]. Web services are considered to be applications that use standard transports, encodings and protocols in order to exchange data and information. They enable computer systems on any platform to communicate in a range of application integration scenarios, within both internal and external organizations. The core architecture of web services, the basic service layer is based on a set of communication standards including the HTTP transport, the XML for representing data format, the SOAP for data exchange, and the WSDL to describe the capabilities of a web service. Additional standards and protocols (WS-* Architecture) are defined over the last few years to facilitate specific functional requirements, such as web services discovery, events, attachments, security, reliable messaging, transactions and management [62],[72].

Nowadays, web services are used largely for the realisation of distributed applications. In spite of the increasing use of web services within internal and external applications, their

functionalities and capabilities (e.g. service interaction, service performance, and overall testability) are becoming key elements in their acceptance and degree of quality.

2.4.2.1 Categories of Web Services

Web services can be grouped into three business-related categories [32]:

1. Business information or device-oriented web services: a business shares information with consumers or other businesses. In this case, the business is using web services to expand its scope. Examples of business informational web services are news streams, weather reports, or stock quotations.
2. Business integration web services: a business provides transactional, "for fee" services to its customers. In this case, the business becomes part of a global network of value-added suppliers that can be used to conduct commerce. Examples of business integration web services include, bid and auction e-marketplaces, reservation systems, and credit checking Business-to-Consumer (B2C) website, across multi B2C systems.
3. Business process externalization web services: a business differentiates itself from its competition through the creation of a global value chain. In this case, the business uses web services to dynamically integrate its processes. An example of a business process externalization web service is the association between different companies to combine manufacturing, assembly, wholesale distribution and retail sales of a particular product, Enterprise Resource Planning (EPR), Customer Relationship Management (CRM) as well as application system integration [93].

2.4.2.2 Web Services' Quality of Services

A service Quality of Service (QoS) description can be published using a variety of mechanisms. These various mechanisms provide different capabilities depending on how dynamic the application using the service is intended to be. The service description may be published to multiple service registries using several different mechanisms. The simplest case is a direct publish, which means that the web service provider sends the service description directly to the service requestor. This can be accomplished using an e-mail attachment, an FTP site, or even Compact Disc, Read-only-Memory (CDROM) distribution. Direct publish can occur after two business partners have agreed on terms of doing e-business over the web, or after fees have been paid by the service requestor for access to the service [58].

2.4.2.3 Web Service Protocol Stack (WS-* Architecture)

There are many standards and protocols that aim to define web service specifications. Moreover, a growing enthusiasm for these standards and protocols is evident from the industry, as they are becoming aware of the key advantages of implementing SOA systems. These standards consist of a collection of norms and protocols, which are open standards that include web transport protocols such as HTTP, FTP and SMTP. The fundamental technologies of the web services model are XML, Messaging and Metadata. Within the web services protocol stack, standards and protocols evolve, merge or become irrelevant.

2.4.3 Extending SOA Design

The dynamic nature of SOA systems present new challenges to the design, development and testing phases. Consumers of SOA systems need to be assured that the services or components will not fail and will return responses quickly.

Currently, the basic generic SOA web services design of service layer do not address concerns of designing and implementing such systems, for example, services management, services composition, and QoS properties that apply to all components in a SOA system. Such concerns can be addressed by extending the SOA design to support such service capabilities. The extended SOA extends and adds new SOA design layers of more advanced service operations protocols, standards and rules. In this way, SOA systems can offer huge advantages in capabilities such as service management by service orchestration and intelligent synchronisation and asynchronisation routing, services provisioning and QoS guarantees such as integrity, reliability and security of messages [64].

The work of Papazoglou and Hevel [64] has presented SOA as an extended model with three logical SOA operation layers, providing a horizontal separation view of the different operations which are involved in these layers, while the vertical view indicates service characteristics that cut across all three layers. The logical separation of operations is based on the need to separate basic service capabilities provided by the conventional SOA operations from more advanced ones, which are needed for composing services, also based on the need to distinguish between the operations for composing services from that of the management of services (see Figure 2.4).

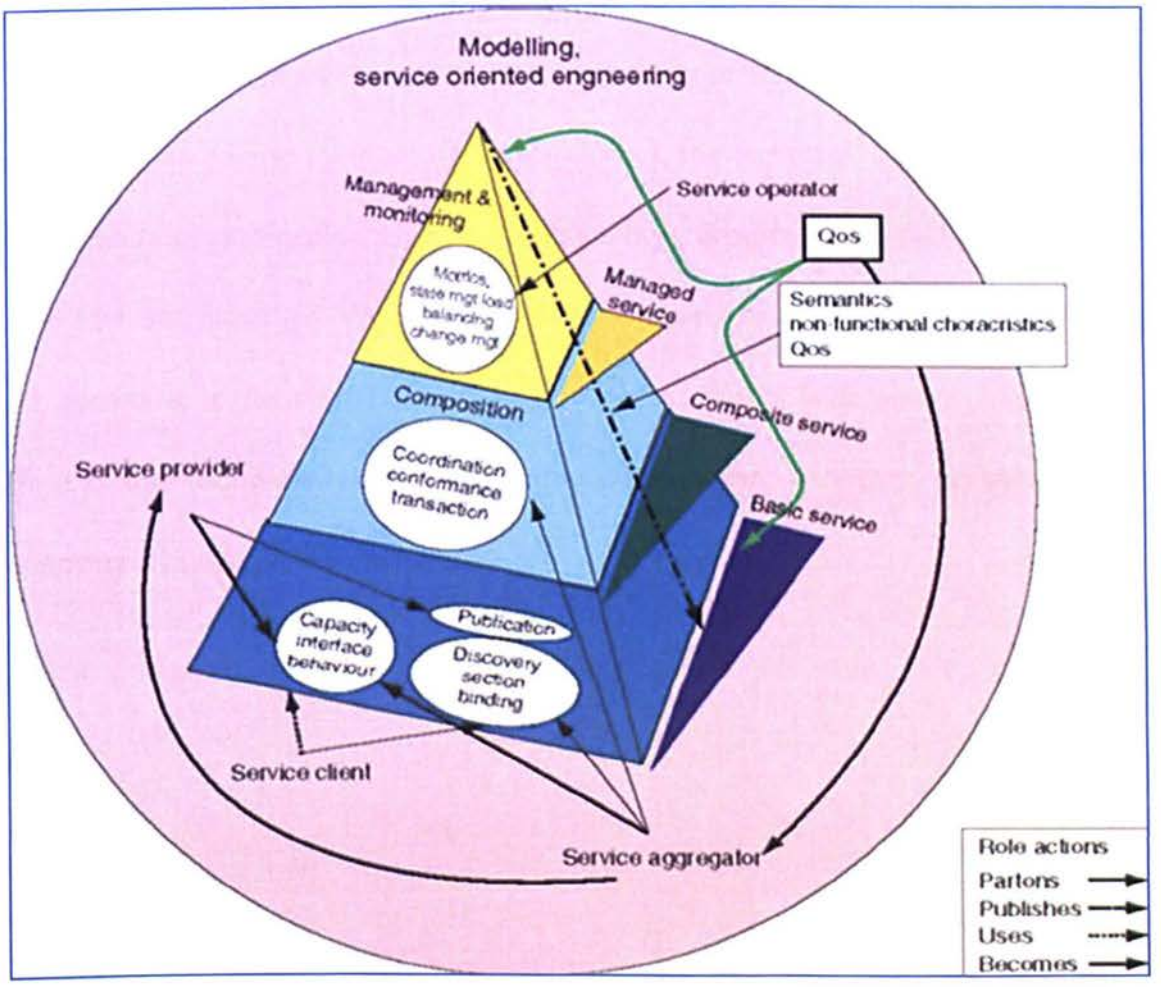

Figure 2.4. SOA design extended layers [64]

While SOAP and HTTP are sufficient for interoperable XML messaging, and WSDL is sufficient to communicate what messages are required between service requestor and service provider, more is needed to cover the full range of requirements for SOA systems. To fully support e-business, extensions are needed for security, reliable messaging, quality of service and management for each layer of the web services protocol stack [49]. Additional requirements for web services infrastructure include support for service context, conversations and activities, intermediaries, portal integration and service flow management.

2.4.3.1 Extending SOA Design to WS-* Architecture

In order to understand and to improve the testability of SOA, which lead to designing a suitable automated and standard-based framework, the technical role and characteristics of the standards and protocols of the web services protocol stack (WS-* Architecture) within SOA was classified. We also classified the elements of the web services protocol stack according to their relationships with different logical layers across the extended SOA, i.e. the foundation layer, the composition layer, and the management and monitoring layer as depicted in Figure 2.4.

Standard/Protocol	Technical Service Role	Service Characteristics	Web Service Layer (according to figure 2.4)
XML	Messaging	Core functional	Foundation
XML Schema			
SOAP			
MTOM			
Web Services Attachments			
WSDL	Description		
UDDI			
WS-Policy			
WS-I			
WS-Addressing	Reliable messaging	Non functional (QoS)	Composition and Coordination
WS-Reliability			
WS-Atomic Transaction	Transactions		
WS-Coordination			
WS-Coordination	Business Processes		
WS-Choreography			
PBEL			
PBMN			
WS-Security	Security		
WS SecurityPolicy			
WS-Trust			
WS-Federation			
WSDM	Management/Monitoring		Management/Monitoring

Table 2.1. The web services protocol stack in relation to the extended SOA

Table 2.1 provides a high-level grouping view of web service standards and protocols which were republished by architecture industrial standards, for example, the Open Group, OASIS and OMG standards organizations. The elements of the web services protocol stack were classified by their relationships to core functional and non-functional (QoS) characteristics. For the core functional characteristics which are required for basic connectivity mechanisms (describe, publish, interact) [30], the standards are: XML, Simple Object Access Protocol (SOAP1.1-2), WSDL, Message Transmission Optimization Mechanism (MTOM), XML Schema, Web Services Addressing (WS-Addressing), Universal Description, Discovery and Integration (UDDI), and Web Services Interoperability Basic Profile (WS-I) [2]. For service composition and for quality of service, several standards have been proposed in the extended SOA system, most notably the Web Services Business Process Execution Language (WS-BPEL) for service composition, Web Services Coordination (WS-Coordination), Choreography Description Language (WS-CDL) and Web Services Transaction (WS-Transaction) to support robust service interactions.

Also the Web Services Security (WS-Security), and Web Services Reliable Messaging (WS-ReliableMessaging) for supporting meaningful business interactions. The descriptive capabilities of WSDL are enhanced by the Web Services Policy Framework (WS-Policy), which extends WSDL to allow encoding and include attachments. Along with the adaption of SOA principles and web services protocol stack, there are relationships and dependencies between each web services protocol stack standard and protocol components (see Figure 2.5)[13].

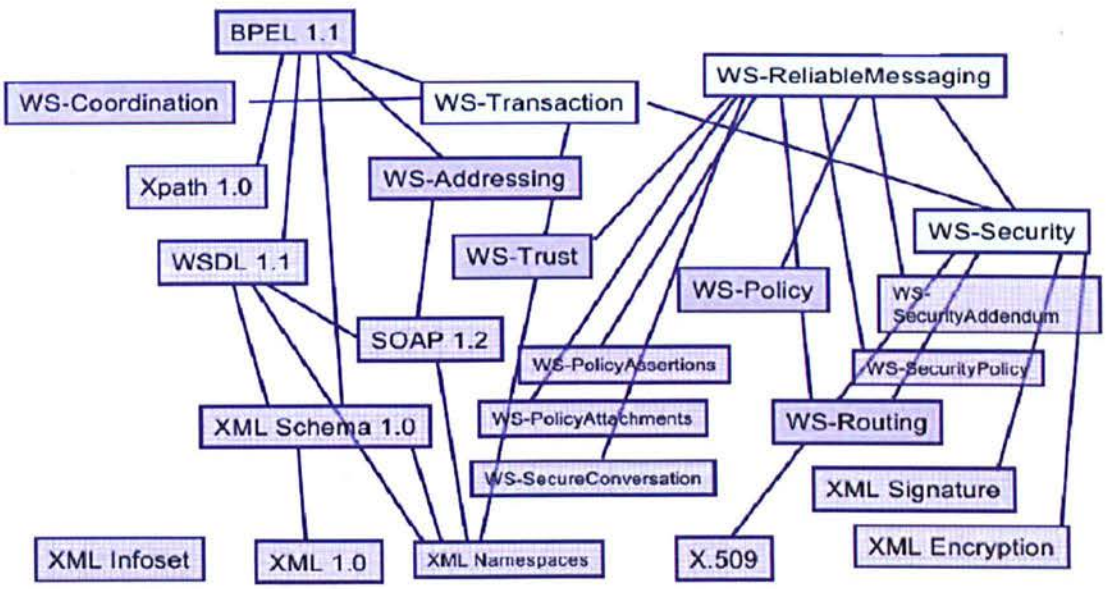

Figure 2.5. Web services protocol stack protocol components relationships [13]

Web services protocol stack covers the full range of requirements for SOA systems. Nowadays, web services are used to a great extent for the realisation of distributed applications. In spite of the increasing use of web services within internal and external applications, their functionalities and capabilities (e.g. service interaction, service performance, and overall testability) are becoming key elements in their acceptance and degree of quality.

2.4.4 Generic Software Testability

The definition of "testability" according to IEEE standards is the degree to which a system or component facilitates the establishment of testing criteria and the performance of tests, determining whether those criteria have been met beforehand and to what degree [88]. Given the context of the above definition, the concept of software testability extends to the approaches and tools which are able to provide adequate testability in software testing, which allows establishment of a feasible and effective software testing. Software testing mainly depends on what the user can see, control and observe using any test sets in any system test level or phase. Noting that, software with an acceptable testability degree ensures that test scripts are executed and satisfactory test coverage is applied. Furthermore, most of the defects should be uncovered and fixed before the product is released [21]. To this extent as mentioned, any software testability is built upon two vital elements: (1) control and (2) observation. To extend the idea, a cost-effective test approach relies on how the systems under test can be better controlled (e.g. making invocations of the services, or setting internal variables, or simulating and changing the execution environments, and so on). Moreover, it also relies on how the systems under test can be better observed (e.g. observing how the system reacts and behaves in response to a test request or input). In addition, it relies on the elements which are related to the combination of the above two elements (e.g. test cases which are dependent on the results of inputs and also on the observation) [95].

2.4.4.1 Generic SOA Testability Problems

Generally, in SOA systems the testability issues are due to [41],[45],[71],[18],[68]:

1. Lack of observation; SOA services need to be invoked, integrated, and monitored within consumers' sites. This hinders testers from verifying and validating the internal and external behaviours of the services in terms of their operational behaviours, input-outputs parameters (test oracles). Moreover, the only information made available for service consumers by a service is the service description in XML-based format, this prevents White-Box testing and Mutation-testing approaches which require access source code of the service.
2. Lack of control; SOA services are run physically independently under the control of the provider. This hinders testers and service consumers from controlling the services operational behaviours, feature customization, and installation and deployment. This also prevents regression testing of the system by consumers due to an inability to decide on the test strategy for a new or the updated version of a service.

Furthermore, looking at testability problems from practical test implementation aspect, we can define the primary differences between SOA testing and traditional application testing as follows [45]:

3. Lack of software artifacts.
4. Lack of control over test executions.
5. Lack of methods of observing system behavior.
6. Problems with testing service compositions, dynamic discovery and invocation capabilities of SOA.

Moreover, examining testability problems from a common software design aspect, the traditional application-centric design focuses on the user interface (Presentation) layer, the process logic (Business) layer, and the data (Resource) layer.

Primarily, the reference architecture of SOA systems is categorised into three main architectures [89]:

1. Business Architecture.
2. Infrastructure Architecture.
3. Information and Data Architecture.

In addition, considering systems heterogeneous implementation, another two layers have been added [47]:

4. Business Process layer to manage the process steps.
5. Integration layer to manage the interoperability of the applications provided by the services.

Thus, many testing methods and approaches that are implemented for supporting SOA experience unique challenges due to the former testability problems within organisations deploying these systems.

One study [2] indicated that organisations at workplaces are conducting several different approaches to solving testability problems, e.g. combining automation in the testing lab and applying changes to the processes at an organisational level. Furthermore, organisations are trying to increase the involvement and communication of the business users in all phases of the software life cycle. Also organisations are now emphasising that quality is not something that has to be required at the end of the software life cycle, but should be considered as a trait that spans horizontally across the business processes. As

indicated in another study [13], the most important missing piece of building SOA systems is the top-down approach, which considers testing disciplines throughout the software life cycle and also encourages the project team to continually determine how they would successfully test the system based on SOA [43]. Another study [123] describes the common blind spots in SOA testing that testers are experiencing in real-life scenarios. The blind spots focus on the areas of performance, security, SOAP, WSDL, and interoperability. The paper concludes that: "through collaboration with development teams, with growing understanding of web services technologies, and with comprehensive SOA testing tools, SOA testers can ensure that the SOA testing blind spots are reduced if not eliminated". Furthermore, most research reported in the literature comes from low-level techniques and implementation details [107], and there is a need for more formal and disciplined implementation of these standards and protocols [53],[49],[108]. Much research, which deals with run-time testing is correspond to the experiments implemented in lab and available for exploitation; however, it's not yet used in real industry environment and its transformation to the support industry is an awkward and uneasy task [91],[95] .This mainly can be resolved using specific protocols of communication and collaboration between the different end points [18],[49].

2.5 Conclusion

In this chapter, in order to establish testability design knowledge and to identify SOA testability problems and consequently improve the testability degree, which leads to designing a suitable automated and standard-based framework a comprehensive investigation is conducted to understand SOA technologies, principles, standards and protocols. In addition, a wide analysis is carried out to breakdown and classify the

technical role and characteristics of the standards and protocols of the web services protocol stack within SOA implementations. Moreover, an extended classification is concluded to classify the elements of the web services protocol stack according to its relationships with different logical layers across the extended SOA. Many testing methods and approaches that are implemented for supporting SOA are experiencing unique challenges due to the testability problems within organisations deploying these systems. The chapter concludes that there is a need to adhere to SOA standards and protocols, as they will progress SOA implementations beyond basic connectivity mechanisms between service providers, integrators and consumers to a services run-time composition mechanism, which supports external process (business-to-business) integration and cross-internal (process-to-process) integration.

Chapter 3 – SOA Testing Approaches and Tools

3.1 Introduction

In the research related to SOA systems testability, several initiatives were conducted with the intention of providing testing approaches and frameworks that support the IT industry which experiences unique challenges deploying and testing these systems[36],[76],[11],[68]. In general, in the area of SOA testing, one faces the problem that common traditional functional and non-functional testing can be very crucial, extremely time consuming and error-prone. Having research through which to view the literature shows there is less research on how SOA services spread around several different computing locations can be tested.

Furthermore, in relation to SOA systems testing coverage, many SOA systems defect and shortcoming reliability issues stem from the fact of poor specification of functional requirements coverage. Typically, SOA system test cases are designed manually by a test engineer with no particular test coverage criteria in mind. As consequence, the typical test case suite covers only about 60 percent of functionality [118]. On the other hand, generally, SOA test automation has gained ground, and accordingly there many tools for automated testing on the market. However, most of these tools simply automate tests that are manually designed. Hence, if the design of a particular test is flawed, or if the suite of tests does not provide full coverage, the test automation offers limited value.

In this chapter research on SOA testing criteria will be considered. What has been done so far in terms of SOA testability knowledge including approaches, frameworks, and tools

will be presented. These related research tools will then be evaluated and classified according to relevant testability methods and strategies.

3.2 Testability Challenges

The adoption of SOA has changed the architecture of computing systems where it caused many changes in the process of designing, building, and using the systems. Furthermore, it has affected traditional testing processes due to several implications in the testability of a system based on SOA. In essence, implementing the important features of SOA e.g. the reusability and interoperability brings up many testability issues. Moreover, the dynamic discovery and invocation methods like runtime binding or dynamic composing between services, allow a flexible and dynamic method of composing the SOA services just at execution time. In this instance, this restricts the ability to test SOA systems beforehand, as the context of any SOA service is often unknown only at run-time. In addition, it restricts testing real-time synchronisation and asynchronisation messages between the different SOA providers and consumers within heterogonous environments. Therefore, a successful SOA test coverage may realise some or all of these benefits and features depending on the quality and requirements' relevance in the SOA implementation designs. However, these implementations pose problems for SOA testers to predict and foresee possible requirements and test usage scenarios between SOA services providers and consumers.

Thus, the idea of testing is challenged by the SOA unique features; primarily it is the distinct SOA testability knowledge that overlaps with research in relation to the SOA unique features e.g., the automated SOA services discovery in heterogonous and runtime binding invocation environments, which imply that the actual services and consumer

configurations which are involved in the service-client invocation are unknown only at execution time. Hence, this limited testability issue of SOA systems appeals reconsidering and redesigning the current traditional and automated testing approaches and to invents new testing approaches and frameworks.

3.3 Current Testing Approaches and Tools

3.3.1 Background

In recent years, traditional manual and automated testing methodologies have been applied to SOA systems, particularly by adapting combinations of unit, integration, system and regression testing methods [36],[76],[27],[68],[17]. These testing methods are normally based on test cases and oracles, which would usually be constructed manually based on tester experience, or automatically generated according to system specifications using test case generation tools. Yet, testing of SOA systems on account of their specific characteristics is more complex than traditional testing methods allow, indeed testing of SOA systems continues to become more complex and challenging. Therefore, the traditional functional and non-functional testing approaches are no longer sufficient. Nowadays, the dynamic composition verification, validation and monitoring in real-time and at run-time environment are new and critical pieces of the overall testing strategy [27],[115],[108]. Added to the mix, are service trustworthiness and robustness, such as integration, performance and security testing, which rely upon run-time testing of QoS parameters of individual services, which largely depend on business process profiling. Still, most of the research approaches for testing SOA systems either extend the traditional testing methods, or attempt to develop new frameworks or approaches by addressing specific issues during the testing procedure [38],[36],[95],[120]. Numerous

contributions have been presented in the literature, primarily in the areas of unit testing, regression testing, integration testing, SOA services orchestration testing, and testing of non-functional properties. We reviewed related tools and frameworks. Based on that, the current testing frameworks, approaches, and tools for testing SOA systems can be categorised according to three testing methods and approaches:

1. Traditional software testing methods.
2. Adaptive SOA systems testing.
3. Testing and evaluation based on SOA systems standards.

The subsequent sub-sections provide descriptions through many SOA testing approaches, tools and frameworks presented in the literature within each category.

3.3.1.1 Testing Based on Traditional Software Methods

The purpose of SOA functional and non-functional testing is to ensure that applications are designed and function as expected, which are usually performed on service components which construct the SOA system. There are common traditional functional and non-functional testing tools, which can be applied in the software life phases of SOA systems, such as unit (white-box), system, integration, performance, compliance, interoperability, and security testing. However, unit testing techniques require access to a SOA services component's code, which is not a practical test as they are required to run in clients' sites. There are many commercial products for SOA functional and non-functional testing, for example Parasoft SOAtest [73] which is an automated commercial web services testing tool which can support WSDL validation, client server unit testing, and functional testing. Other commercial products can be used for unit testing, some of which include LISA WS-testing [60], Borland SilkPerformer SOA edition [14], jBlitz [50],

and Agitar Agitator [4]. A set of test cases, both positive and negative, can also be generated by SOAPSonar from a WSDL file [123]. The tool can discover defects in the target web services using Gray box testing, in an effort to push Black box testing towards White box testing. Prasanth et al. [117] proposed an automated utility called ATU (Automated Testing Utility) for functional testing of the core web services standards; WSDL and SOAP, by using Sax (Simple API for XML) for parsing WSDL file, Apache Axis tool to generate SOAP stubs , and using manual input parameters to invoke web services methods. Chan et al. [20] proposed a metamorphic testing framework for supporting unit testing, and follow-up action is made automatically to generate integration test cases. The framework applies a dynamic services integration test autonomously. Non-functional, e.g. load testing can be done interactively by SoapUI [91] and TestMaker [75]. These are open source tools for testing web services which can also be used for scalability and performance testing. Referring to this type of testing, Schieferdecker et al. [83] presented an automated web services testing approach. The approach is a flexible test framework for web services testing, using Testing and Test Control Notation (TTCN-3). TTCN-3 is an international standardised testing language, which can be applied to a variety of application domains and testing types [108]. TTCN-3 can provide service interactions, functional testing, and load testing with flexible and various test configurations. A conversion tool is used to automate the translation of XML data to TTCN-3 notation based on the specification of the web service. In this way test beds can be generated directly.

3.3.1.2 Adaptive SOA Software Testing

These approaches can handle a specific SOA systems feature, or particular function testing, or they can offer an alternative presentation to a traditional testing method, e.g. solutions focusing on dynamic composition of SOA systems. Generally, these approaches are designed to make the tests shorter and more accurate. Mattsson et al. [57] propose an alternative method to the traditional testing method, by creating complementary roles to take responsibility for integrating the business processes and ensuring their interoperability and service quality. These include a Business Process Integrator, which is responsible for monitoring the interoperability and business process components integration. Another role is Business Process Tester, which is responsible for testing the overall business processes and validating them against their functional and non-functional requirements. An automated composite testing approach in a simulated environment is described [9] by Bartolini et al. The approach is called Service-Oriented Coverage Testing (SOCT) and is composed of a service provider called (TCov) which sits between the integrator and the service provider. TCov monitors test execution, and provides results to the service integrator through a published web service testing interface. An approach using an automated online test tool is introduced by Dustdar et al. [82] for testing SOA and Component-Based Distributed systems. The work presents a real life prototype called SITT (Service Integration Test Tool). SITT is designed in such a way that it can also be used for monitoring SOA endpoints by using test daemons. Bertolin [10] presented the PUPPET environment for the automatic generation of stubs. The tools simulate the behaviour of the external services, which invoke a service under test environment. The approach implements knowledge modelling techniques to generate

test cases from the functional specification, and from the SLA of the composed services. Zhu [45] discusses the testability of SOA just before the invocation while the SOA components are in the operational environment. The author proposes the creation of a Testing Service (Automating Test Services), which can be either provided by the same vendor of the functional services, or by a third party (an external Test Service).

A model-driven testing approach is presented by Lee [120] based on an MDA reference model (Model Driven Architecture). The author proposes the implementation of a business-centric SOA test framework, for testing SOA systems based on an MDA reference model in a business-centric way. A test harness (called BOSET) is presented to simulate business process functionality, so that it behaves as a BPM (Business Process Management) method, and thus simultaneously tests the overall system performance. Here, the test harness can execute business process, perform tests and create metric values concurrently. Another approach which is also regarded as adaptive SOA testing is the Data-Driven test using an ontology reasoning approach. This approach specifies precisely the data that will be exchanged among service entities, in order to design test scenarios in a methodical way. Offutt et al. [67] presented a knowledge modelling technique for testing the interaction between pairs of web services using a Data Perturbation approach. The technique is based on modifying values in the captured requested messages, then sending the messages to the service consumers, and obtaining the test results by analysing the values in the responded messages. Another Data-Driven test tool called WS-TAXI [19] has been developed to provide a framework for generating web service test suites. In this framework, the testers combine test coverage provided by a SoapUI tool with WS-TAXI to generate data-driven test cases based on the XML schema.

An agent-based approach was proposed by Lennon [59]; the approach uses agents to determine the different elements of composite web services, and to ensure they are complying with the overall clients' SLAs. The author considers these agents as a form of middleware between clients and service providers. Moreover, Taranti et al. [71] proposed using a MAS for testing SOA systems. In this research, the agents are deployed as external testing simulators, which act as providers and as consumers within the SOA system. These simulators were developed to test the critical functionalities and qualities of ship monitoring systems in order to improve safety and security at sea. The system is used as an interface for the integration testing phase, and also to obtain performance metrics for validating non-functional requirements of the system. The system was developed by the Brazilian Navy. Lazarou et al. [99] highlighted the research which covers the deployment of software agents to support software testing. The authors concluded that the proposed work is theoretical, and there is an important need for tools to be developed to verify and validate SOA systems, mainly through the use of test agents. The authors add that current approaches have researched the prototype levels, and implementing a richer variety of test agents is advisable. In particular, a highly significant approach would be to employ deeper intelligence techniques, e.g. machine learning, consecutively as this will enhance agent capabilities.

3.3.1.3 Testing Based on SOA Systems Standards

These are the frameworks and approaches which can handle the testability of SOA systems, according to architecture industrial standards, e.g. the Open Group, OASIS and OMG standards organizations. These frameworks require understanding of the various emerging standards from W3C for testing SOA systems, from the perspectives of service

provider, consumer, and services registry (UDDI). The most important of these emerging standards concerned with the service composition, are the Web Ontology Language for Services (OWLS), the Choreography, and the Orchestration standards [115]. Web service orchestration is defined as a specification which aims to standardise integration logic and long-running processes across web service systems. Some popular open source products, such as SoapUI, JUnit and TestMaker, can be used for testing orchestration of services which are involved in the communications between business services and application services [120]. A framework utilising architecture industrial standards is presented by Dai et al. [42]. The framework uses model checking for web service compositions, which is based on the BPEL4WS standard. Another piece of research on testing service composition is presented by Bucchiarone et al. [17]. The work highlights the status of testing techniques using standard approaches for web service composition, and discusses strategies for doing integration testing from orchestrations and similarly from a choreography perspective. Canfora and Penta [36],[38] addressed the challenges of SOA testing from the perspective of applying an non-traditional testing approach. The authors discussed testing a service discovery feature, focusing on run-time discovery of services in open environments, with late binding availability. The authors propose combining testing and monitoring as an alternative method of traditional testing for SOA systems. They propose testing the SOA system by continuous self-checking and monitoring of the services during execution time, while using complementary roles for testing and run-time monitoring. Bertolino et al. [11] also suggested active auditing and testing of the services by the web services registry. Dung Cao et al. [28] propose a passive system testing approach for verifying observable traces of messages sequences between web services

providers and consumers using set of real-timed constraints. Passive testing is usually used as a monitoring technique for detecting and reporting system errors when active testing method cannot be performed. Active testing allows testers to interact directly and find problems with the system under test [28], while passive testing is better suited as a troubleshooting approach to identify source of problems on a web services site after they have occurred [3].

3.3.2 Current SOA Testability Summary and Evaluation

Having navigated through many SOA testing approaches in the literature which involved manual and automated testing methods, these approaches can be defined as a combination of unit, integration, system and regression testing. The test cases and criteria can be established, and updated according to test beds. Test cases are constructed manually based on testers' familiarity and experience of the SOA system under test, or generated automatically from source code, or using Model-Driven Architecture (MDA) testing [119],[9]. Many of these approaches and tools are applied over SOA systems in typical simulated environments [10],[73],[99],[94],[119]. Other online testing and monitoring approaches are presented along with other SOA testing approaches [9],[76]. The approaches propose testing the SOA system in relation to dynamic binding compositions at run time. This is done by the continuous self-checking of SOA systems, and monitoring the services during execution time, while using complementary roles for testing and run-time monitoring, in the process of testing the overall SOA systems.

Testing SOA systems is attempting to be a more automated and self-managed process [30]. Various automated approaches have been proposed and developed, which can

support the generation of test cases from system specifications based on web service ontology languages [27],[114]. The test cases and criteria can be established, and updated according to test beds. Test cases are generally created from service specifications, service contracts, or service log files, and also from user inputs. Test cases are generated also from communication messages and events between services and clients in a typical SOA architecture, and via typical simulated test environments.

Moreover, industry and software engineering have not kept up with the emerging SOA standards and protocols, most importantly for dealing with run-time services' trustworthiness, and overall quality assurance [58],[115],[41],[56],[25].

Moreover, several frameworks utilised automated testing using the MAS techniques in the development and testing phases. The agents can create and execute different test types in each phase of the software life cycle. Several frameworks cooperate as multi-agents working together in parallel in order to minimise testing resources, time and costs. The multi-agent can perform positive testing to verify the required functionalities, and they can perform negative testing to ensure the robustness of the systems by using specified inputs from internal sources or invocation. However, most of these MAS test systems are used as typical simulators in traditional user-defined test environments. These simulators are easy to use although they ultimately fail to capture most of the SOA real-time environment characteristics [113]. Moreover, simulating the behaviour of external SOA services to invoke a service manually under a test lab, or in vitro [22] is not a valid test due to the essential idea that there are simply too many possible SOA system configurations and options to test in variant client environments, ultimately this requires wide test environment scenarios.

Additionally, testing SOA systems' lack of sophisticated tools provide better testability knowledge and deeper intelligence, this includes MAS, or any other automated system [59]. These systems need to be implemented in realistic and automated testing approaches with the ability to apply the testability knowledge about SOA system under test within accurate and specified test environments, and generate test cases and analyse the outcome as measured by effective testability criteria.

Thus, as testing SOA systems is more sophisticated than traditional testing methods, which is basically because SOA systems are distributed applications with numerous run-time behaviours, and using testing automation tools implemented in traditional simulated environments cannot accommodate the fundamental characteristic of the dynamicity and adaptability of SOA systems.

Moreover, the ideal approach to enabling an effective testability degree of SOA systems should be based on combining test simulators as offline testing with online testing and monitoring for validating and verifying the system, and service's trustworthiness. The system under test can be supported by realistic, controlled, and intellectual capabilities of MAS frameworks. MAS will enable full automated offline-online testing, and monitoring systems which are more practical and accurate than manual testing. MAS supported by intelligent reliability can play a key role in solving much of the automation, monitoring and quality assurance problems, because they can support proper tests and early defect leak detection.

3.4 Conclusion

This chapter reports several investigations into means of improving the testability of SOA systems. Nevertheless, many of the approaches and tools do not provide support for

input or output (test oracles) data selection, for which they still rely on the human tester's intervention.

Moreover, up to now, most of the research into SOA systems testing is still theoretically based on formal methods, such as model checking and FSM (Finite State Machine) testing methods. Furthermore, most research into SOA systems test execution and verification reported in the literature comes from low-level techniques and implementation details.

In addition, industry and academia have not kept up with the emerging SOA system standards and protocols, most significantly with those for dealing with run-time service trustworthiness and overall quality assurance.

Thus, some research dealing with run-time testing is available for exploitation, though its transformation to support industry is a difficult, if not impractical task. A salient point in the literature on automated MAS and other automated testing systems is the lack of sophisticated tools that would provide SOA systems testability with deeper levels of intelligence and prudence. In general, such tools and approaches could automatically derive skeletons of SOA systems test cases and provide support for their execution with continuous monitoring in real-time environments and result analysis. In the next chapter, the proposed framework which will address many of the testability issues in SOA will be presented.

Chapter 4 – An Intelligent Framework for SOA Testability

4.1 Introduction

Based on the results of the analysis and substantive problems of the current testability state of SOA systems in Chapter 2, there is a general acknowledgement that the concept is limited due to a lack of observation and control of the testing processes. Furthermore, based on facts from the literature review in Chapter 3, where an evaluation was presented, it was observed that a numerous SOA testing approaches, frameworks, and tools are proposed to bring structured solutions for improve SOA testability. However, the existing approaches and frameworks are still considered inadequate to satisfy the need for improving the testability of SOA systems. For that reason, these approaches and frameworks are still theoretical and their transformation to support the industry is a difficult, if not impractical task. Thus, this chapter proposes an automated and systematic monitoring SOA system testability framework supported by the ML technique and based on the knowledge discovery and data mining of protocols and standard requirements and test coverage analysis. The framework provides a practical solution to the testability problems in SOA systems by automatically establishing testability links between the prerequisites of intelligent knowledge of SOA testability and the system under test requirement and test coverage analysis.

This chapter describes the proposed framework for SOA system testability which aims to combine existing computational techniques and methods for resolving the problems of testability. The chapter starts by presenting a suitable approach for improving testability in the context of integration of ML in the testability in which the new approach has

evolved. The implemented framework is described together with all the required modules and functional flow processes within the prototype framework. Finally, the findings are summarised in the conclusion.

4.2 SOA Testability Status Recapitulation

The proposed framework acknowledges the use of offline-online testing and monitoring to meet the lack of the SOA system's testability factors and improve the testability degree of SOA systems. The proposed approach also acknowledges that the use of automated and systematic testing software has now become a vital in the software industry. Consequently, employing the intelligent reliability approach can be a very useful part of SOA automated testing systems and would increase SOA testability.

However, the proposed framework approach also acknowledges the fact that systematic requirements based on the functional and non-functional testing approach of SOA systems are widely unexplored [53],[108],[41],[56],[114],[49].

The proposed framework approach also acknowledges the fact that SOA standards and protocols define various aspects of standards and protocols, such as WS-* Architecture which consists of a collection of such which are open standards specifications, for example the transport protocol, document types, security requirements and transactional properties. Hence, in order to guarantee interoperability and QoS between SOA service providers and consumers, organisations that play certain roles in collaboration have to support these standards or protocol specifications. If a standard or a protocol specification does not cover certain aspects, these organisations have to agree on them in order to achieve interoperability and QoS of their applications.

Moreover, with SOA, consistent monitoring of testing processes is required, this includes input-out data, and subsequent analysis of test results to determine causes of the defects and recommend solutions. Such a testing approach requires testability knowledge across all heterogeneous environments and across all SOA services providers and consumers, so that a proper fix can be applied to the SOA system under test.

For this reason, the proposed framework enables overall SOA nature and principles to be determined, which can then be identified based on a requirement of SOA standards and protocol. The proposed framework provides a practical solution to the problems by automatically establishing testability links between prerequisites of SOA testability, requirements coverage, test case coverage, dynamic testing and test monitoring, and test coverage analysis. It is of interest to note that the focus of this research work is analysing and improving testing systems as actual and applicable deliverables for the industry, and accordingly increasing the testability of systems based on SOA principles, rather than theoretical refinement of SOA testability design or testing processes.

4.3 Framework Conceptual Design

The conceptual design of the proposed framework is to utilise automation while using two defect detection techniques: the dynamic and static analysis techniques and a combination of both. Control and observation within testing processes are initiated by using a static analysis technique to evaluate the system under test specifications according to the WS-* Architecture. As pointed out in previous section, adapting WS-* Architecture is considered the key enabler for deploying web services in the industry. By adapting WS-* Architecture, the industry will also be able to adhere to the principles of SOA. These principles give SOA systems an explicit goal of defining the modular

technology stack for supporting and resolving the communication and collaboration between different parties within SOA implementations. The framework provides defect finding using directed test generation based on learning properties from test cycles and monitoring outputs. The process is supported by a Learning and Decision Making (LDM) module for learning, reasoning and decision-making process (see Figure 4.1). The LDM utilises knowledge about the SOA system under test and problem domains, and produces the appropriate KB feedback for the requested modules within the system. The static test and dynamic test analysis techniques can be performed in numerous ways based on the properties of the service under test and on service specification. For example, a static test analysis can be used to verify service specifications through web service interfaces like WSDL, PBEL, and BPMN documents, whereas general dynamic testing analysis aims to identify general defects in the entire system. The dynamic testing will be performed by running test agents as simulators of service clients for consuming the SOA services. Test procedures will be described in test case scenarios which define the system's input-output and run-time environment settings.

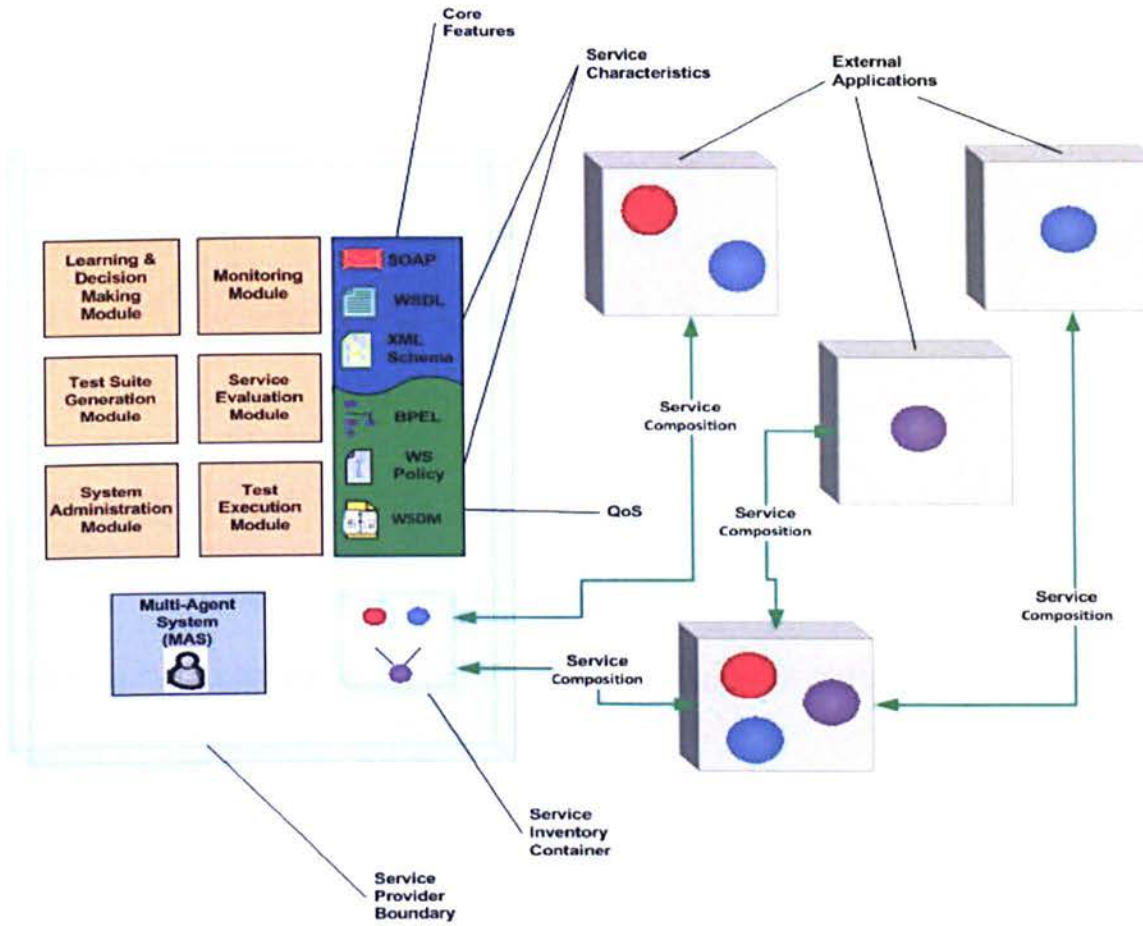

Figure 4.1. An overview conceptual design diagram of the proposed framework architecture

The system properties learned from the LDM module are fed into the test case generation system. The tests from this system are executed and their outputs added to the testing data database. Each test cycle uses the LDM KB feedback as a guide to explore new test inputs that were not explored in the previous iterations. As the test cases are executed and the outputs added to the testing data database, the coverage of the tests is enhanced, and test cases which are expected to expose defects in the system are increased. However, many errors may still remain in the system that needs to be exposed by formulating appropriate test cases for run-time properties, for example service composition testing. As compensation for this, run-time operational monitoring explores

the paths and input data values that have not been covered by testing cycles. This can support automated testing and early defect leak detection, and expose these remaining defects. Also it effectively handles positive and negative tests during test executions, and during run-time operational conditions. In general, using KB feedback from LDM module will support the learning and decision making concept, and will be able to produce effective quality level testing.

4.3.1 Functional Requirements Specification

This section highlights the functional requirements and relationships between different modules in the proposed approach. Figure 4.2 illustrates the functional requirements specification of the proposed architecture.

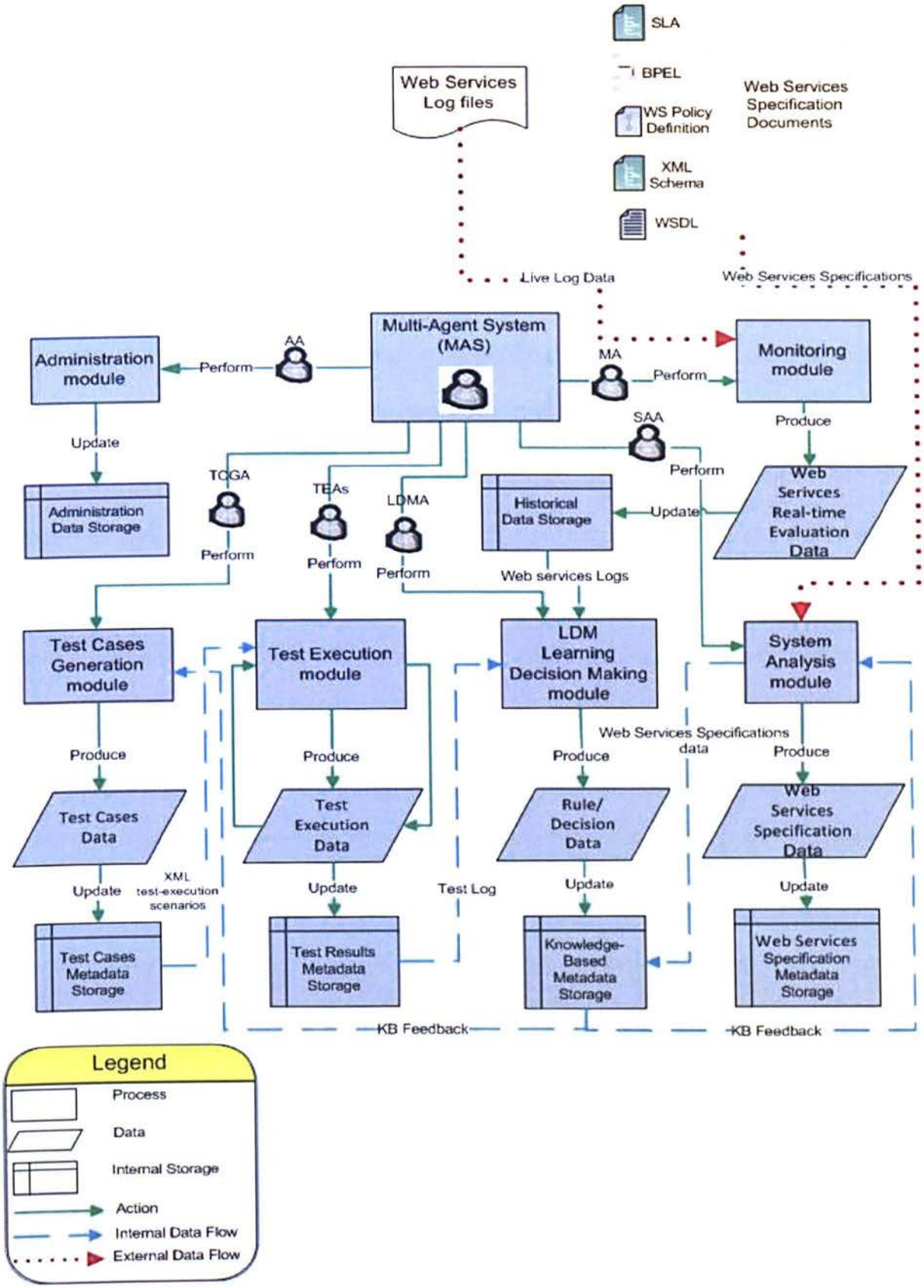

Figure 4.2. An overall view of the function relationships among the modules of the framework

As mentioned in Section 4.2, a static test analysis can be used to verify service specifications through web service interfaces, for example WSDL specifications. The system analysis module employs system analysis agent (SAA) to perform static analysis on service specifications, SAA analyses the SOA system to determine system properties and behaviour by extracting the required data from WSDL structure. KB feedback will be provided by the Learning and Decision Making Agent (LDMA) using a form of ML algorithm. A decision-tree learning algorithm can be used for data mining the properties and behaviours of the web service under test, and for populating the data set and validating the discovered patterns. The decision tree learning algorithm classifies the properties and the behaviour of the web service according to mapping and matching to WS-* Architecture in an attempt to find possible requirement mismatches of the web service properties and behaviours, which will allow for achieving and maintaining adherence to SOA principles.

The system properties and behaviour obtained from the system analysis process will be saved in the service metadata database. This data can then be used to generate test case abstractions through the test case generation process. The test case generation module employs a Test Case Generation Agent (TCGA) to generate baseline test cases, runtime environments and execution scripts which are supported by intelligent learning and decision making from the LDM module. The LDM module combines learning, reasoning, and decision- making capabilities to explore web service characteristics, i.e. input-output properties and parameters, QoS elements, and other operation conditions and behaviours. Additionally, the defects and inconsistent execution patterns that were captured during test cycles could form learning facts or premises in the intelligent

learning and decision-making process through the LDM module. LDM module processes functional requirements and test coverage criteria by verifying and analysing the results of test executions, and makes decisions that correspond to it and the targeted test environment, these tests which will cause the service under test exercises positive or negative behaviours.

The Test Execution module will deploy Test Execution Agents (TEAs) to carry out test simulation tasks. The test execution tasks could include for example, service invocation under given a simulation task by a test scenario. Test Execution module then performs test result analysis and inputs the results in metadata database. LDM then analyses the output data obtained from test execution procedures, and verify the condition of the test cases determining whether the test cases have met the initial test objectives. Otherwise, new test cases need to be generated, targeting the service under test to exercise the behaviours which are outside the executed test cases. The test outputs at the end of each test cycle will be joined to the earlier data inputs in the metadata database, this data will be used by the LDM module in the process of exploring and exposing advance defects. The LDM module provides automated KB feedback as a guide to explore the test outputs that were obtained from the test cycles. Hence, this concept supports the implementation of the intelligent automated testing and defect detection approach.

The Monitoring module deploys Monitoring Agent (MA) to perform operational monitoring. This process will allow for learning useful facts from operational monitoring, and exposing the errors which may still remain in the service under test through offline testing, for example, run-time service composition by service consumers within live heterogeneous environments. MA monitors web service log files in real-time, and checks

for the errors caused by the interaction between service requester and web service provider. MA then will transform these errors to a specific format and then save them in the historical database. The error logs in the historical database then can be used by the LDM module as defect logs for analysis whether the different components of the web service have met the requirements of the run-time operational standards. These defects then may need to be exposed by the creation of appropriate new test cases. This process effectively handles positive and negative tests during online operational conditions.

On the other hand, the administration module uses the administration agent (AA) to control certain aspects of test agents' allocation, schedules, and administration such as, MAS executions and synchronisation in real-time conditions. AA administrates and observes the simulation tasks in real-time and makes the MAS environment controllable and flexible. AA also will help the modularity, scalability, and interactivity between the test agents, through administration in a distributed MAS environment.

The subsequent sub-sections provide a description of each module in the proposed approach, which can to be read in connection with the accompanying module's flowchart diagram.

4.3.1.1 System Analysis (SA) Module

The System Analysis module systematically captures and classifies the core functional and QoS standards and specifications of the web services under test. The captured data will then be used in the process of generating test cases, and in the processes of test verification and validation. Furthermore, as mentioned previously, the focus of this research is on including the WS-* Architecture within the testing process, to specifically enable effective automated testing and increase the testability of SOA systems. The

System Analysis module analyses the web services specifications using the WSDL and possible BPEL and BPMN documents. These XML files can be used to define the necessary standards and the protocol implementation requirements by means of a formal contract between the web services consumer and the web services under test. Figure 4.3 describes the cross-functional and data flow within the system analysis processes. The flowchart diagram exemplifies the technical details of the activities of the underlying design of this module.

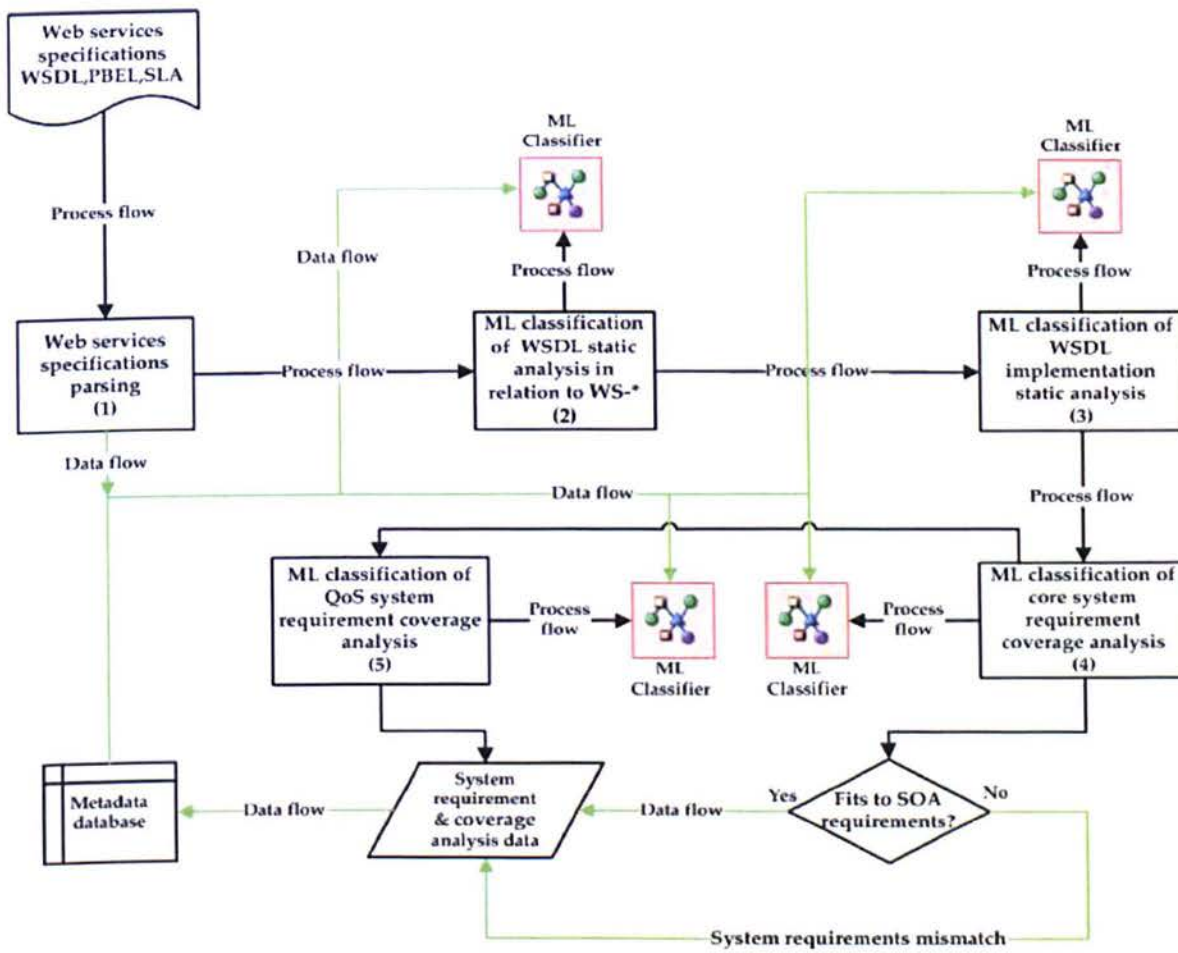

Figure 4.3. Functional process flowchart of the system analysis module

The System Analysis Agent (SAA) activates by receiving an Agent Communication Language (ACL) message with "start SA" content from AA.

Accordingly, the SAA first parses the available WSDL and BPEL documents and transforms them into a structured DOM (Document Object Model) tree (see Step 1 in Figure 4.3). Subsequently , the WSDL Static classifier (see step 2 in figure 4.3)—a machine-learning classifier and an implementation part of LDM module is used to generate SOA web services under formal test requirement and specifications data according to the core functional and non-functional (QoS) standards and specifications within the web services protocol stack.

Next in Step 3 in Figure 4.3, another machine-learning classifier of the LDM module—the WSDL Implementation Static classifier is implemented to support SAA and generates the conventional core implementation parameters and variables of web services under test such as, operating methods, agreement binding, message types, service description, service publication and discovery based on core functional standards and specifications in the WS-* Architecture. Generally, this data is required to achieve successful invocations and interactions between web service providers and consumers.

Next in Step 4 in Figure 4.3, another machine-learning classifier of the LDM module to support SAA, the System Requirement-Coverage classifier uses the outcome of data mining process knowledge from previous steps to define and classify a structural machine-learned requirement-coverage metric. The metric will be then used to guide the test-data generation and to derive a test suite. The System Requirement-Coverage classifier process data mining to identify and assess the core functional characteristics of the web services under test. The Requirement-Coverage classifier uses a mismatch checklist to identify the possible requirements mismatches of the service core components according to the WS-* Architecture. Thus, the SAA with the support of LDM

module will determine if the system under test requirements is valid according to outcome of the data mining of the core requirement-coverage analysis.

Similarly, in Step 5 in Figure 4.3, the QoS System Requirement-Coverage classifier generates QoS requirement-coverage analysis metrics as structural and machine-learned coverage criteria of the web services under test, and according to the non-functional-QoS standards and specifications within the WS-* Architecture. The QoS System Requirement-Coverage classifier uses the outcome of the data mining process by the WSDL Static classifier (Step 2 - Figure 4.3) which contains the mapping references of the QoS for web services under test specifications to the QoS protocols and standards within the WS-* Architecture. This includes messaging, addressing, security, reliability, transactions and so on. These protocols and standards support simple and complex QoS requirement message and behaviour patterns between web services provider and consumer. For example, if a web service being tested is advertising QoS as WS-Policy assertions then a set of behaviours concerning those properties can be used in conjunction with the web services provider and the consumer messaging communication patterns.

Accordingly, a training dataset is used to set a classification model of the possible QoS classification for each specified QoS protocols and standards within the web services protocols stack. Each class then set to tuple—sample data patterns which are assumed to belong to each QoS class mapped to syntaxes, semantic and rule are provided in the training dataset. A predictive machine-learning algorithm—the J48 decision-tree classifier [61], is used to learn the training dataset, modelling a classifier model for the QoS System Requirement-Coverage classifier. Eventually, the QoS System Requirement-Coverage classifier searches for syntaxes of the QoS behaviour properties and values for the

possible QoS data mining. The process of data mining determines the setting of the web services client's test-harness environment that will be needed in the test case generation and test-execution of each operation.

4.3.1.2 Test Cases Generation (TCG) Module

The Test Cases Generation module generates systematic automated test cases supported by the machine-learning method. The generated test cases range from a formal test cycle process to scheduled ad-hoc test cases. The process of test case generation adapts the static reasoning analysis technique based on system input-output and QoS properties obtained from the web services under test specification data mining tasks from the former system analysis process. The test cases generation module covers all the combinations of the WSDL elements by performing the systematic generation of black-box test cases. For this reason, the approach applies the Category Partition (CP) technique [97] to the WSDL. CP provides an approach to identify the relevant input-output parameters and QoS conditions and enables their values to be combined into datasets of categories and partitions. Once these datasets are available, a machine-learning algorithm is used to learn the rules that relate to the generating sets of (category, choice, and constraints). The indicated constraints will be produced using the Equivalent Classification Partitioning (ECP) technique [65], which can then be used for generating abstract test cases. Practically, the identifying individual operation of the web services under test can be carried out separately. Figure 4.4 illustrates a flowchart diagram of the cross-functional and data flow of the Test Cases Generation module.

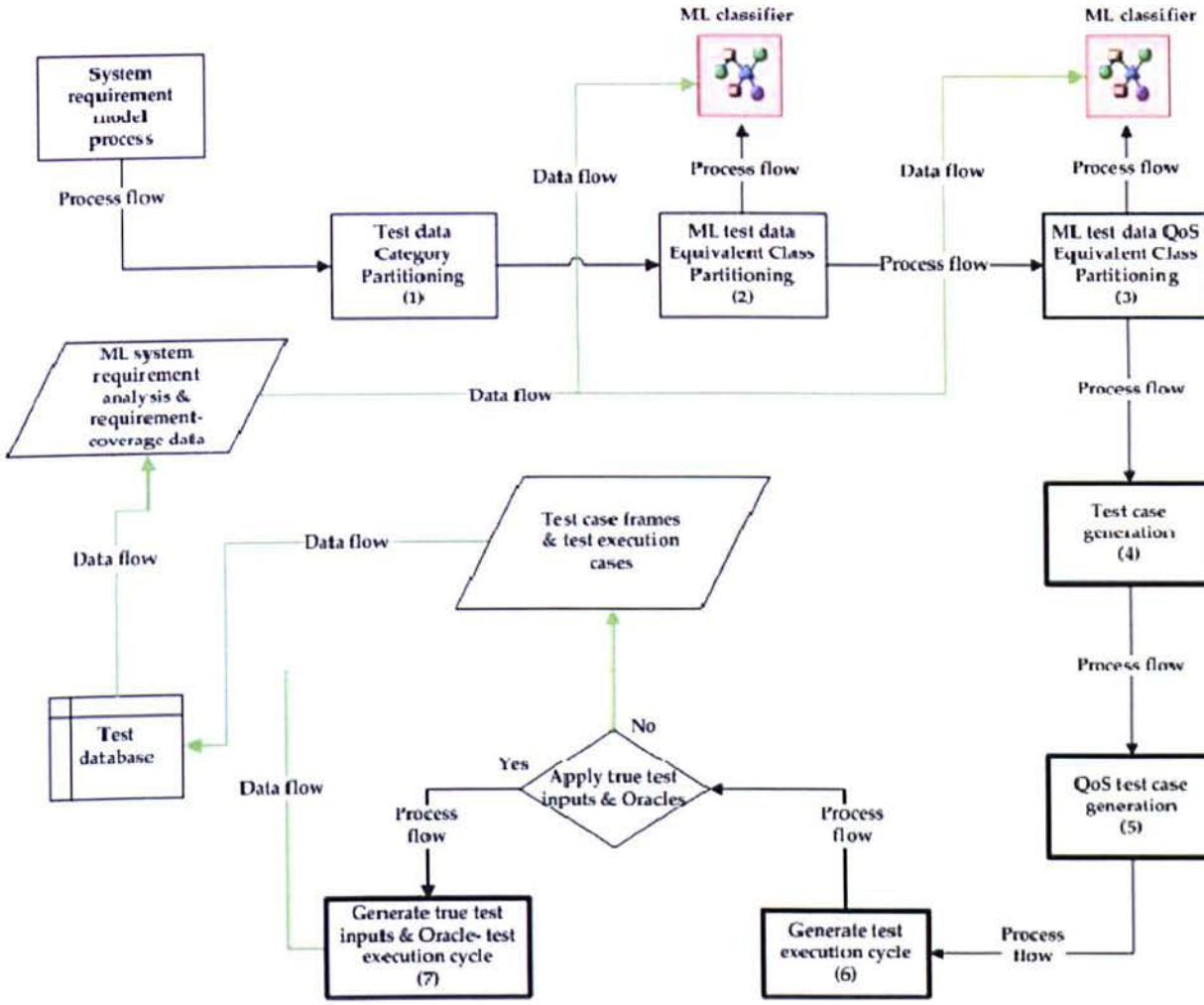

Figure 4.4. Functional process and data flow flowchart diagram of the Test Cases Generation module

The underlying idea of using these techniques is to develop a systematic and automated static analysis and structural formal specification-based test cases generation approach, by using a MAS. On the other hand, the Learning and Decision Making module can produce correct classification decisions based on training datasets to capture new data tests.

Initially, the test case generation starts with the CP procedure (Step 1 in Figure 4.4) by capturing and identifying the data from the results of data mining of the core and QoS requirements specification of the web services under test, which already is saved in the metadata database. For each of the web services' WSDL, the process identifies and

classifies the web services data into categories of operations, input-output data type properties, and QoS characteristic values. The possible default-explicit input-output data type properties are further constrained by restrictions called XML facets, which are used to define the acceptable values of the XML elements and attributes [116]. The possible WSDL data-type input-output facets can be classified into input and output equivalent value ranges using the ECP technique (Step 2 and 3 in Figure 4.4). The different combinations of input data classifications are selected and arranged into test sequences. The Test Cases Generation module then writes abstract frames of the test core and QoS test cases (Step 4 and 5 in Figure 4.4) including the data result from CP and ECP processes. Finally, the Test Cases Generation module transforms test case frames into XML test-execution scenarios (Step 6 and 7 in Figure 4.4), adding the actual input test values, expected result (Test Oracles), the test ID, and the test cycle number. For the actual input test and oracles values, two approaches are adopted: (a) values can be picked from an associated test inputs-oracles table, or (b) generated randomly according to the input value conditions that are derived from the ECP step. The generating test-execution XML files can include test inputs and oracles manually as required by the user. Accordingly, the Test Cases Generation module configured as "Apply true oracle?" option (see Figure 4.4) for including true test inputs and oracle assertions (the exact values of expected test results). The true test inputs and oracle values are specified in the test_inputs_oracles table in the metadata database. Hence, the XML test case files are updated with the exact value assertions by the "Generate true test inputs & oracle- test execution cycle" (Step 7 in Figure 4.4). The true test inputs and oracles are determined

by manual analysis. Actually, this is the only manual effort required during the testing process.

4.3.1.3 Test Execution (TE) Module

The Test-Execution module utilises TEAs for serving as entities which are capable of carrying out automated testing processes. The process includes test execution simulation tasks, performing server response analysis, and feeding the test executions results into the test database. The test simulation tasks could be for example, exploring inconsistent execution conditions based on run-time monitoring events, or scheduled test harness to demonstrate defect finding in accordance with the SOA principles. The typical objectives of a test harness are; (1) to automate the testing process, (2) to execute the test suites, and (3) to generate the associated test execution results logging.

TEAs simulate test service consumers and execute tests according to inputs, outputs, and service behaviour test data through the test cases. TEAs behave as service invocation stimuli either based on test case formation or based on run-time operational events, which are received from the administration module. The test simulation outputs at the end of each test cycle will be joined to the earlier inputs in the test database, which will be used by the LDM module in the learning, reasoning, and decision-making processes within the monitoring process. Figure 4.5 demonstrates the workflow of the activities within the test execution module.

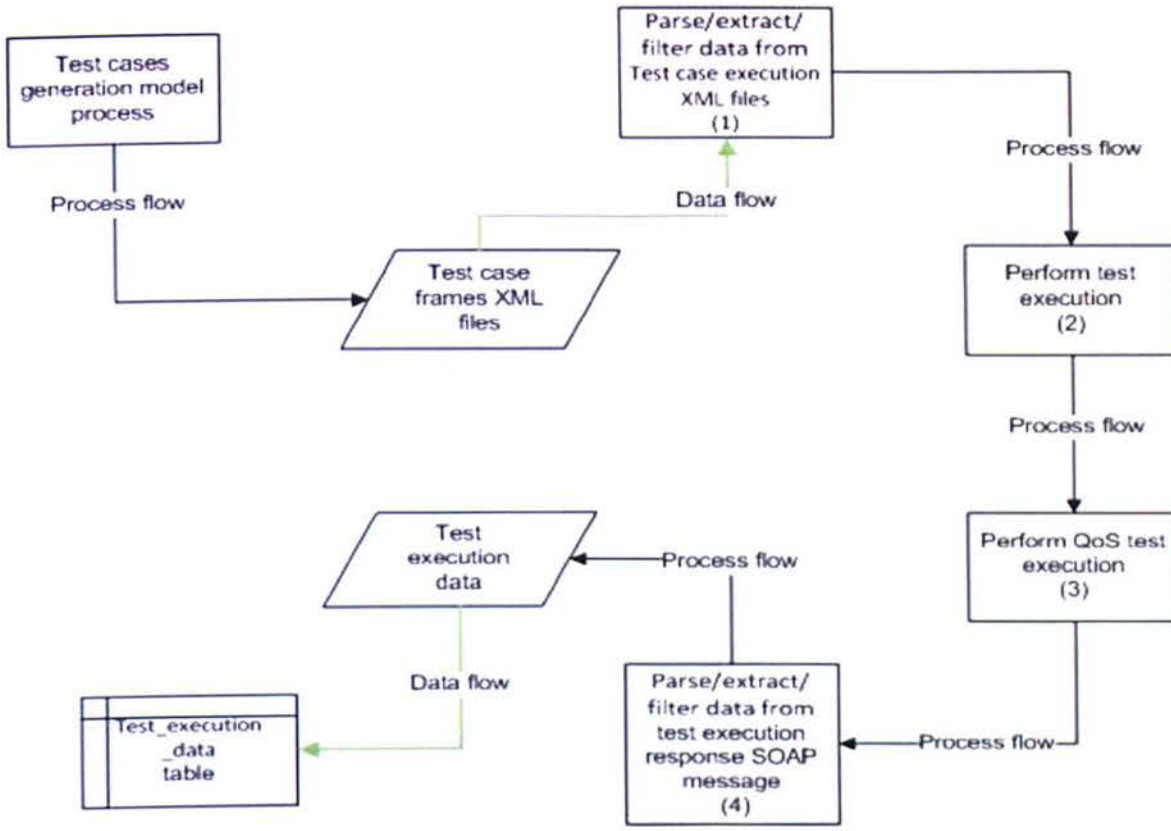

Figure 4.5. Functional process and data flowchart of TE module

As mentioned earlier, the LDMA also searches for any applicable QoS behaviours mapping the data to the web services protocol stack, this data is included in the XML test cases as part of the headers, which are afterward used by the TEAs for test-harness implantation which include test environment setup, test-execution, and test-execution response report of the web services test client. Through Step 1 in Figure 4.5, the TEA parses and identifies the test data in the test execution XML file; the data includes the web services name, the target name space, the end-point URL, the port name, the operation to be tested, the unique Test ID, the HTTP and the request test message body content which includes the test inputs parameters. Moreover, the identified test data includes the environment settings, which are required for implementing the test harness of the web services test client or consumer at the time of creation of the SOAP message communication dispatcher.

Through Step 2 and Step 3 in Figure 4.5, TEA is ready to initiate the dynamic testing processes in order to validate the system and QoS requirements. The TEA executes XML test-execution files using the unique test ID to identify and match test execution results for each test execution simulation task. Simultaneously, the TEA waits for the web services response message send from the web services under test in real-time. The TEA generates test execution reports, by transforming the SOAP message to a data string for parsing and extracting the test response data (Step 4 in Figure 4.5). The response data includes the Test ID, the response Body, and the message content result. The TEA updates the test_execation_data table in the test database with the test execution result for each case according to its unique test ID number. The test simulation outputs at the end of each test cycle are used by the LDM module in the learning, reasoning, and decision making process, throughout the test validation procedure within the monitoring process in the next stage.

4.3.1.4 Learning and Decision Making (LDM) Module

The Learning and Decision Making module makes use of LDMA to combine learning, reasoning, and decision-making abilities in generating test-case scenarios and exploring the properties of the test-harness implementation, such as test-execution environment and the test-execution results for the web services tested. Predictive machine-learning algorithms and implementation parts of the Weka system [61] are used to learn the datasets that relate to the web services protocol stack. The learning system is trained with the training datasets using classifiers.

As mentioned in Sub-sections 4.3.1.1 and 4.3.1.2, the LDMA classifies and maps data properties and values of methods, input-output data-type of the web services under test,

and any core functional standard and specification, for example the web services name, the target name space, the end-point URL, the port name, the operation to be tested, the output method name, the output expected data-type name, and the output element type. Also, the LDMA searches for QoS behaviour properties and values in the web services specification metadata database, mapping service specifications to the web services protocol stack, from which new test-execution environment classification cases may be predicted. The TEAs use this data to set the test-execution environment of the web services test client or consumer when the SOAP message communication dispatcher is created and the web services are invoked. Additionally, the LDM module verifies and analyses the test-execution results, and obtains the decisions corresponding to the web services protocol stack, realised from the web services specifications. Defects and inconsistent test-execution patterns could arise from the facts or premises of the testing process. The LDMA will have access to the test database, which contains defective data, tracking logs, system properties, and QoS characteristics of the web services under test, according to the web services protocols stack which includes messaging, addressing, security, reliability, transactions, metadata exchange, etc.

In general, the feedback from the LDMA will support the learning and decision making concept, and will be capable of producing effective quality-level testing. In accordance with the expected test outputs, the LDMA can analyse the output data of test executions of the baseline test cases, classify their condition, and determine whether they have meet the test objectives. If not, the generation of the new test cycle is targeted to cause the system to execute the re-testing process. Figure 4.6 shows the activity diagram of the LDMA supporting classification and data mining of other modules in the framework.

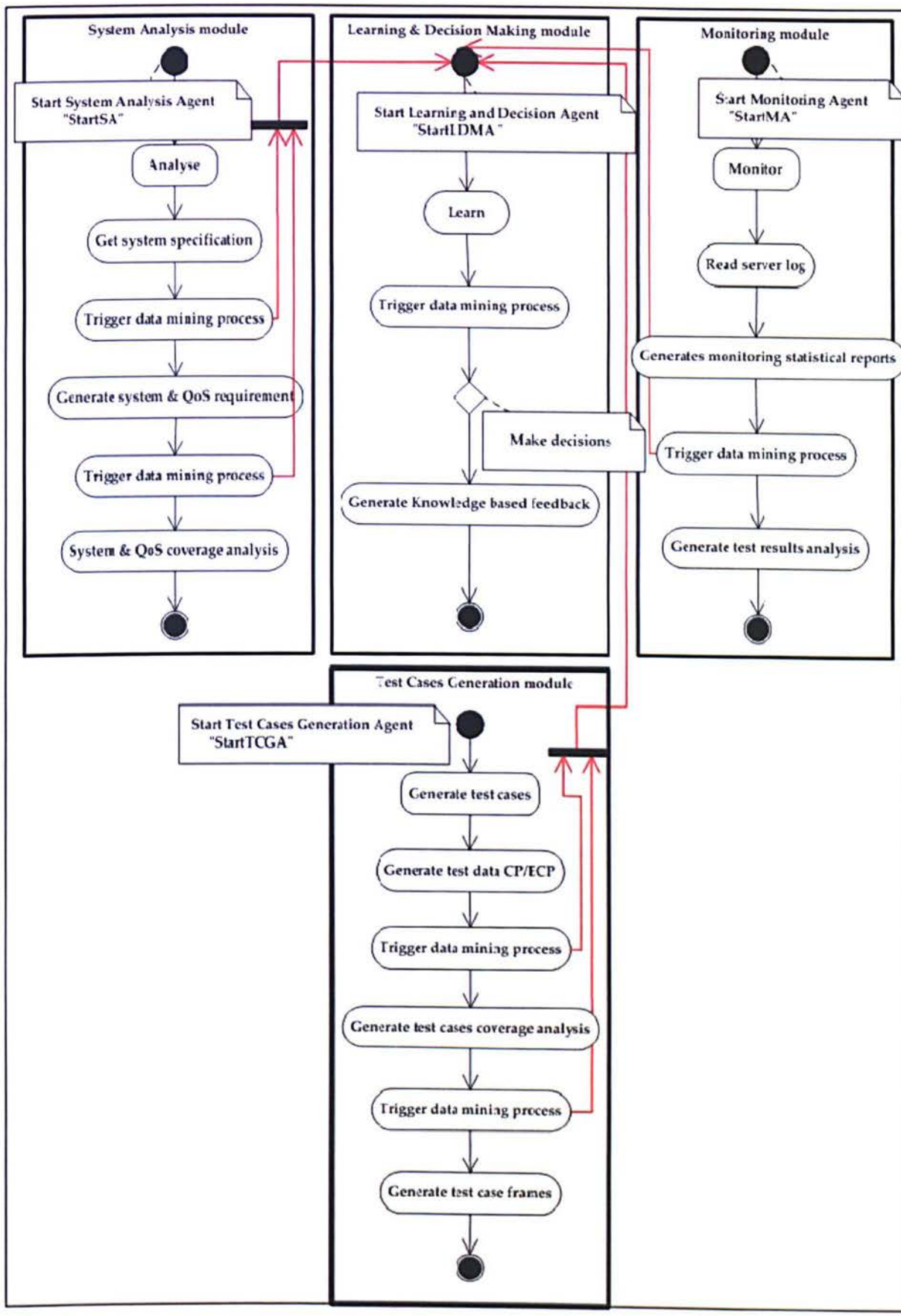

Figure 4.6. The activity diagram of the LDMA supporting classification and data mining of other module in the framework

4.3.1.5 Monitoring Module

The system Monitor module deploys MA to monitor the web services in the real-time environment. MA monitors and checks service log file for messages occurring throughout interactions between service requesters and the web service under test in active status, MA then reports these messages and transforms them to types of customised messages which can be used by the LDM module, it then saves these messages in the System_Historical_Data database. This data will be considered as observed learning inputs at a monitoring period. The LDM module will analyse and judge whether the components of service meet the requirement of the system operational standards, otherwise it will need to expose new test experiences and extra maintenance effort to be directed. Figure 4.7 shows a flowchart of the monitoring module.

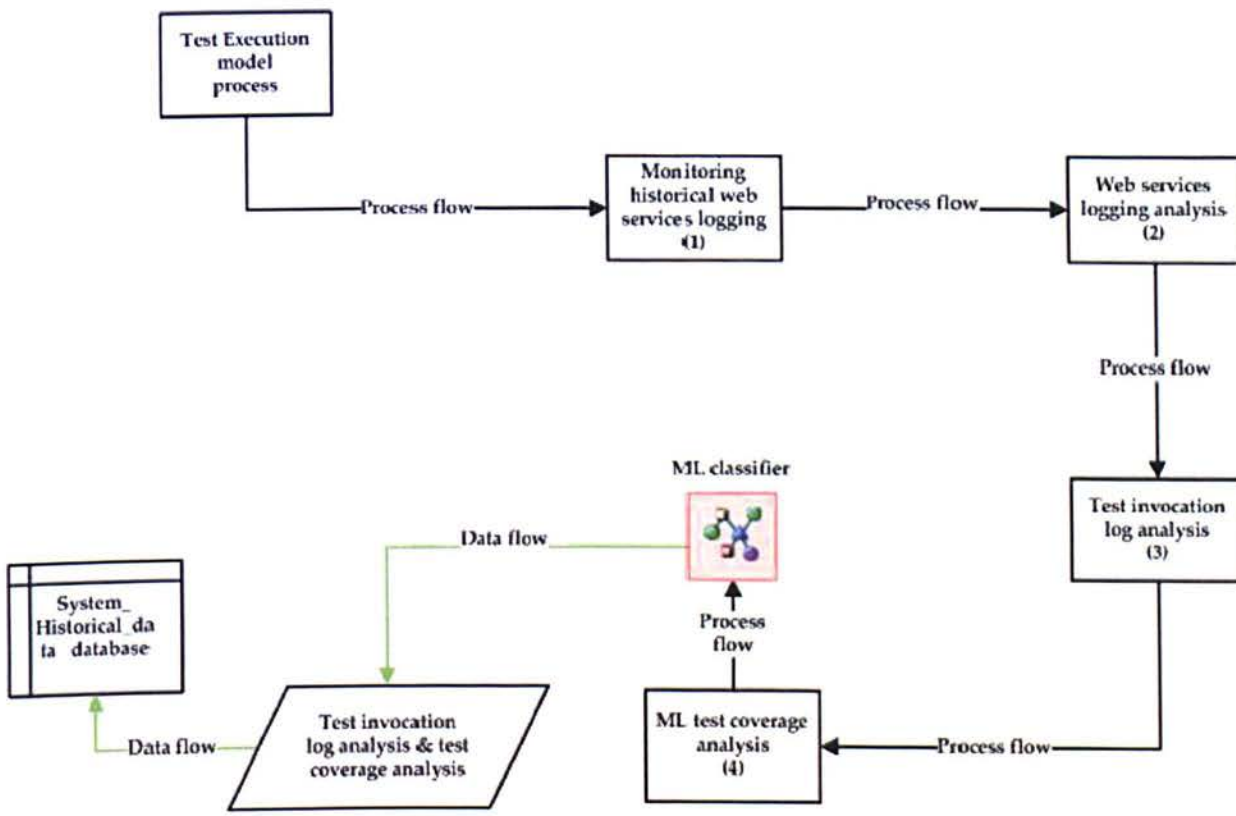

Figure 4.7. A flowchart diagram of the Monitoring module

The monitoring process includes run-time operational monitoring of the web services log file for messages and logs sent between the web services test client and web services in real-time (see Step 1 in Figure 4.7). The MA generates monitoring analysis and statistical reports, including Test ID, Message ID, HTTP Request and Response Body, message content, etc. (see Step 2 and Step 3 in Figure 4.7). The MA updates the System_Historical_Data database with the new results test invocation analysis for further test coverage analyses. The results of the testing data metrics become an input for the trained machine-learning classifier. Using the J48 decision-tree classifier, a new classifier model is trained with the training dataset to validate against the testing dataset (see Step 4 in Figure 4.7). According to the testing results, the LDM module will judge whether the test executions of the test cases meet the expected results and requirements of the QoS system's operational standards.

4.3.1.6 Administration Module

The TEAs will be required to run as scheduled test harness executions, or as concurrent multi-agent executions which make the simulation tasks for a number of test executions within distributed environments. As a result, TEAs may need further instruction from AA in order to execute test cases properly. AA extended the proposed approach infrastructure to run-time infrastructure which supports agents' allocation, schedules, interaction, and synchronisation. AA sends task events based on formal test cases or on given test conditions captured though system monitoring log files which are already saved in the System_Historical_Data database. The AA passes the external events as discrete monitoring events and once an event comes from the monitoring agent, the administration agent will put the event into its waiting queue in the metadata database.

The events in the waiting queue will processed depending on the test task allocation procedure, i.e. if the allocation task procedure is formed as an multi-agent task simulation, any incoming event can be processed as long as there is enough resource, if the system is formed as a single-agent simulation, an incoming event can be only processed until no one else is using the processing resource and so on. Figure 4.8 illustrates the sequence workflow diagram of the events between system administration module and others module in the framework.

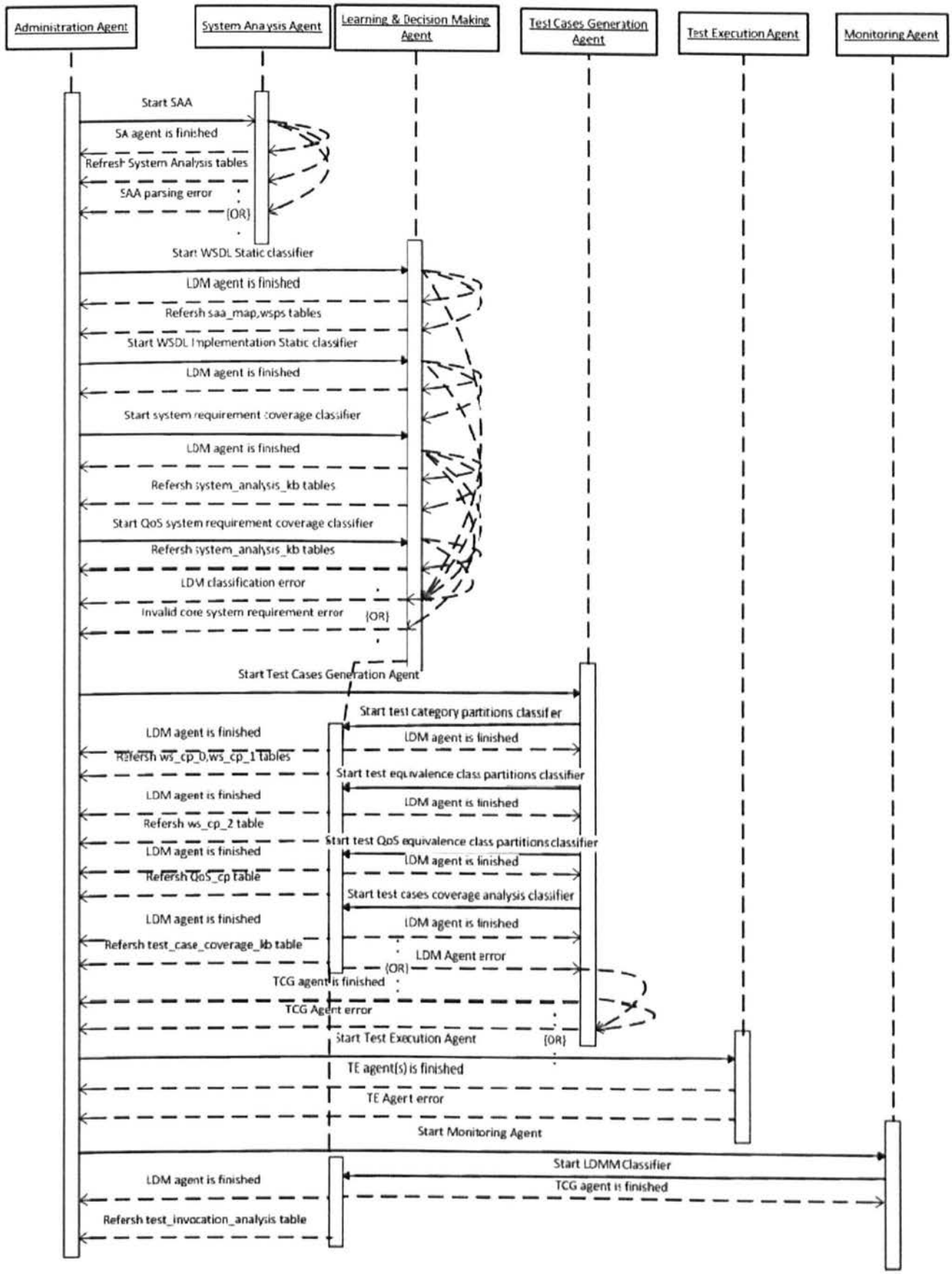

Figure 4.8. The sequence workflow diagram of the events between system administration module and other modules in the framework

4.4 Conclusion

This chapter started by presenting a suitable approach for improving SOA testability, in the context of integrating the machine learning in the SOA testing on which the new framework has evolved. Then, the design of a new framework for SOA testability is described which aims to combine existing computational techniques and methods for resolving the problems of SOA testability. Then, aspects of the implementation of the machine-learning SOA testing framework are elaborated with detailed descriptions together with all the required modules and functional flow processes within the framework.

In summary, in this chapter, the many aspects of the new framework for SOA testability were considered. The approach provides a practical solution based on a functional prototype to resolve the testability problems in SOA by automatically establishing testability links between the prerequisites of intelligent knowledge of SOA testability and system under test requirement and test coverage analysis. A suitable approach was presented for improving SOA testability, in the context of integration of machine-learning in the SOA testability on which the new framework has evolved. The chapter presented the design and implementation of an automatic machine-learning framework by means of a functional prototype implementation. The framework realises the advantages of the MAS approach supported by intelligent reliability, such as preferences, with purely SOA principles and a standard-based approach using the web services protocol stack. This is essential to increase the QoS and level of deployment within both academic and industry sectors.

The underlying idea of using these techniques is to develop an automated testing cases generation module that applies a structural machine-learned core and QoS requirements based specification—an automated black-box test cases generation process by using MAS technique. On the other hand, machine-learned data mining process produces correct classification decisions based on the training dataset applied upon new classification cases. Then, an intelligent test case-coverage analysis supported by machine-learning method is applied to determine that the test suite satisfies the coverage criteria according to the core and QoS requirements specification of the web services under test.

An empirical evaluation of the functional prototype of the framework using practical examples based on the quantitative data analysis of cost-effectiveness will be carried out in Chapter 5, in order to evaluate and prove that significant saving in time and effort and can be achieved by employing the developed framework.

Moreover, a practical case study will be presented in Chapter 6 to test and evaluate the functional prototype of the framework by using it in a real-life business situation. It is also to present the possible integration of the framework architecture in the current SOA computing infrastructure, with aims of confirming the suitability of the proposed architecture within industries, or companies and their strategies.

Chapter 5 – Framework Evaluation

5.1 Introduction

According to the Capability Maturity Model (CMM) by the Software Engineering Institute (SEI), in order to assess and improve the IT architecture, all CMM compliant organizations that have reached the specific level of maturity (Level-4 and above) must have Information Technology (IT) architecture defined, managed, and all quality and performance metrics captured, measured and associated with the IT architecture [39]. Within this context, the primary objective of studying cost-effectiveness metrics of software testing techniques used by frameworks or tools could lead to measuring, comparing and evaluating which lead in order to improve software testing techniques and practices.

An analytical evaluation study has been carried out in order to evaluate cost-effectiveness of the proposed framework exclusively by using the following key factor; test cost, defect detection effectiveness and cost-effectiveness measurements within the testing phase. All of these are considered as primary factors which can produce concrete structures of test framework. The result helps to define high quality degrees of frameworks in a cost-effective manner. However, though intuitively, the empirical analysis of test cost and defect-detection rates are considered the primary parameters for measuring cost-effectiveness of a testing technique, still they consider proportional measures since there are other validity assessment factors. These validity factors can be utilised for cost-effectiveness by comparison with other SOA test frameworks and tools. For example, practically in relation to internal validity factor, the degree level of automation for a testing technique could be determined by comparing it to current

innovative research, or typically comparing it to similar open source and commercial tools. However, depending of the choice made, the results which are obtained from the assessment could in realistic situations represent a lower bound in suboptimal if there is no representative technology or available tools to match the specific level which is encountered in the compared testing techniques. The following subsections will provide analytical evaluation descriptions of the practical evaluation study of the functions of each core module in the proposed framework. Then, in the following sections and sub-sections, the proposed framework will be empirically evaluated and compared to other frameworks and tools for testing SOA systems.

5.2 System Analysis Example

The System Analysis module first generates web services under test requirements and specifications by parsing the WSDL document and transforms them into a structured DOM (Document Object Model) tree. The WSDL data is generated by all data tags according to standard XML syntax in the WSDL document. This includes filtering the main WSDL's tags-sections which are: the wsdl:definitions, the wsdl:types, the wsdl:message, the wsdl:portType, the wsdl:binding, and the wsdl:service tags which are generated based on core functional standards and specifications in the Web Services Protocols Stack. The WSDL data included in these six tags-sections is filtered and the extracted data for each section is inserted correspondingly in six tables (named as WSDL extract tables) within the metadata database. Figure 5.1 shows a screenshot of the six WSDL extract tables including the extracted WSDL data following the System Analysis module parsing process.

WSDL XML parsing

Binding info | PortType info | Types info | Service info | Message info | Definitions info | BPEL

id	definitions	binding	operation	input	body	output
1	definitions	definitions	binding type="tns...	binding type="tns:S...	operation name="getTickerPrice" s...	binding type="tns:StockTickerPrice" bind...
2		binding typ...	binding type="tns:...	binding type="tns:S...	operation name="getTickerPrice" s...	binding type="tns:StockTickerPrice" bind...
3		binding typ...	binding type="tns:...	binding type="tns:S...	operation name="getLastUpdated"...	binding type="tns:StockTickerPrice" bind...
4		binding typ...	binding type="tns:...		operation name="getLastUpdated"...	
5		binding typ...	binding type="tns...		operation name="getChange" soap...	
6			binding type="tns:...		operation name="getChange" soap...	
7			binding type="tns...			
8			binding type="tns:...			
9			binding type="tns...			
10			binding type="tns:...			
11			binding type="tns...			
12			binding type="tns:...			

Figure 5.1. A screenshot of the WSDL extract tables of the SAA parsing process

The WSDL Static classification model learns from the training dataset, which greatly depends on the training data patterns which are presented in the dataset. Considering that the web services protocol stack elements can provide the core functional and QoS training dataset for the WSDL Static classifier, the training data patterns are investigated according to the classification of the technical role and characteristics of the standards and protocols of the web services protocol stack in the extended SOA design layers in Sub-section 2.4.3.1. Each element in the web services protocol stack in the extended SOA design layers is labelled as a categorical class within the training dataset. Each class is then set to tuple—sample data patterns which are assumed to belong to each class, as determined by the class label attribute and rule. Then, the classification algorithm discovers knowledge from the training dataset and constructs a classification model and can be evaluated using the evaluation module. The WSDL Static classifier uses a Naive Bayes classification method [94], which is a simple probabilistic classifier based on Bayes' theorem with strong independence assumption for text mining.

The WSDL Static classifier then performs WSDL text mining of the web services core and QoS protocols and standards which are searched and captured from the WSDL extract data, which new classification cases may be predicted according to their relationship to core and QoS element within the web services protocol stack. The result of classification

is saved in a specific table (named as WSDL_TO_WSPS) as references for mapping core and QoS services specifications to web services protocol stack. Subsequently, the outcome of the above data mining is used in the validation and verification processes which include defining core and QoS requirement-coverage metrics as structural coverage criteria, the structural coverage metrics will be used to guide the test-data generation and to derive a test suite including test execution environments. The test suite will be used during dynamic testing implementation, and then during test operational monitoring and test coverage metrics during the verification stage.

In order to train and build a classification model, the WSDL Implementation Static classifier is trained by data patterns semantics which are investigated and mapped according to the syntaxes in the generic SOA design layers as in Section 2.4.1.1. The generic SOA design layers provide the core functional standards and protocols to achieve a successful invocation and interaction implementation between web service providers and consumers. Hence, these data patterns provide a data set of the request and response messages exchange, the service data, and the signatures of its operations which they need to be extracted from the WSDL elements.

Each element in the web services protocol stack in the extended SOA design layers is labelled as a categorical class within the training dataset. Each class then set to tuple—sample data patterns which are assumed to belong to each class, as determined by the class label attribute and rule. These core functional standards and protocols requirement syntaxes can be discovered and extracted from the six WSDL's sections tags which are following : (1) <definitions> the root WSDL element which declare the namespaces used in the document, (2) <types> element which contains the data types which will be

transmitted, (3) <message> element which contains the information about the messages which will be transmitted , (4) <portType> element which contains the information about the operations that will be supported, (5) <binding> element which contains the information about the means which the messages be transmitted on, and the <service> element which contains the information about the location— URL address of the service. Then, the training data patterns semantics are classified and mapped to the syntaxes of the core functional standards and protocols within the generic SOA design layers.

This data is required to achieve successful invocations and interactions between web service providers and consumers. Collectively, this data is considered the core functional and non-functional (QoS) testing resources of the formal requirement specifications of the web services under test. Accordingly, in order to train and build the classification model of the WSDL Implementation Static classifier, the semantics of the trained data patterns are investigated and mapped according to the syntaxes of the core functional standards and protocols in the WS-* Architecture design layers which provide the core functional standards and protocols to achieve a successful invocation and interaction implementation between web services providers and consumers. Table 5.1 lists the applicable core functional standards and protocols syntaxes of the web services implementation, which are used in the training data patterns. Table 5.1 also lists the technical roles and the mapped WSDL tags syntaxes and attributes for each of the core functional standards and protocols.

Required data	WSDL mapping element	Technical role	WSDL mapping tag and attribute	Searched keyword in WSDL extract table
targetNamespace url	Definition	Must be the root element Declare the namespaces used in the document	<wsdl:definitions name="targetNamespace=""/>	"targetNamespace="
Name of the service	Service	Define the name of the service	<wsdl:service name="">	"service name="
Where the service is located?	Service	Defines the end points (i.e. address) of web service	<wsdlsoap:address location=""/>	"address location="
Port name(s)	Service	Defines one or more binding ports	<wsdl:port binding=""/>	"port binding="
How messages will be transmitted?	Binding	Defines transmission media (e.g. HTTP, FTP and SMTP)	<wsdl:soap:binding transport=""/>	"binding transport="
Message style	Binding	Defines Message format type (e.g. SOAP, REST, XML-RPC)	<wsdl:binding transport="" style=""/>	"binding style="
What is operation input(s)?	PortType	Defines the operation input elements to form a complete one-way or round-trip operation	< wsdl:PortType input wsaw:Action=""/>	"input="
What is operation output(s)?	PortType	Defines the operation output elements to form a complete one-way or round-trip operation	< wsdl:PortType output wsaw:Action =""/>	"output ="
What is message name?	Message	Defines the name of the request/response messages	< wsdl:Message message name= ""/>	"message name="
What is operation name?	PortType	Defines the name of the service operation	< wsdl: PortType operation name=""/>	"operation name="
What are message part names?	Message	Define the message part names elements	< wsdl:Message part name=""/>	"part name=""
What are message part elements?	Message	Defines the message part elements	< wsdl:Message part name="" element=""/>	"part element="
What are simple datatype names?	Types	Defines the simple data type names used by the web service provider and consumer	< wsdl: Types element name =""/>	"element name="
What are complex datatype names?	Types	Defines the complex data type names used by the web service provider and consumer	< wsdl: Types complexType name=""/>	"complextype name="
What is data type value(s)?	Types	Defines the data type values used by the web service provider and consumer	< wsdl: Types complexType name="" type ="" />	"element type="

Table 5.1. List of core requirement assertions and applicable training data patterns of WS-* Architecture

Accordingly, the WS-* Architecture elements provide the core functional training dataset for a machine-learning classifier. Each WS-* Architecture element is labelled as a class within the training dataset. Each class is then set to sample data patterns associated with each class and with a predicted classification rule which is determined by the class label attributes and rule. Using a machine-learning classifier, the LDM module learns from the

training dataset, which greatly depends on the training dataset size. A Naive Bayes classifier [94]—a simple probabilistic classifier based on Bayes' theorem with strong independence assumption for text mining —is used as the WSDL Implementation Static classifier for learning and classification of the WSDL data to the WS-* Architecture elements. The classification algorithm discovers knowledge from the training dataset and constructs a classification model and can be evaluated using the evaluation module. According to the data mapping patterns from the 5.1 table, the WSDL Implementation Static classifier is trained and a training dataset is built.

The WSDL Implementation Static classifier then processes data mining analysis by extracting and classifying the grammar rules, properties abstractions, and concrete elements of the web service under test from the WSDL extract tables in the metadata database. The interest is including the web services name, the target name space, the end-point URL, the port name, the operation to be tested, the output method name, the output expected data-type name, and the output element type. The WSDL Implementation Static classifier then saves the outcome of the data mining process for further use during the validation and verification processes, which include the core SOA principle, standard requirement-coverage, and test coverage measurement analysis.

The WSDL Static classifier then performs WSDL text mining of the web services core and QoS protocols and standards which are searched and captured from the WSDL extract data, which new classification cases may be predicted according to their relationship to core and QoS element within the web services protocol stack. The result of classification is saved in a specific table (named as WSDL_TO_Implement) as references for mapping core and QoS service specifications to web services protocol stack. Subsequently, the

outcome of above data mine is used in the validation and verification processes which includes defining core and QoS requirement-coverage metrics as structural coverage criteria, the structural coverage metrics will be used to guide the test-data generation and to derive a test suite including test execution environments. The test suite will be used during dynamic testing implementation, and then during test operational monitoring and test coverage metrics during the verification stage.

Next another machine-learning classifier of the LDM model to support SAA, the System Requirement-Coverage classifier uses outcome of knowledge of data mining process from previous steps to define and classify a structural machine-learned requirement-coverage metric. The metric will be then used to guide the test-data generation and to derive a test suite. The System Requirement-Coverage classifier process data mining to identify and assess the core functional characteristics of the web services under test. The Requirement-Coverage classifier uses a mismatch checklist to identify the possible requirement mismatches of the service core components according to the web services protocol stack. Thus, the SAA with the support of the LDM model will determine if the system under test requirements is valid according to outcome of the data mining of the core requirement-coverage analysis. Figure 5.2 shows a highlighted section of the decision tree structure of the training dataset evaluation of the WSDL Implementation Static classifier, which is based on core functional standard and specification requirement-coverage.

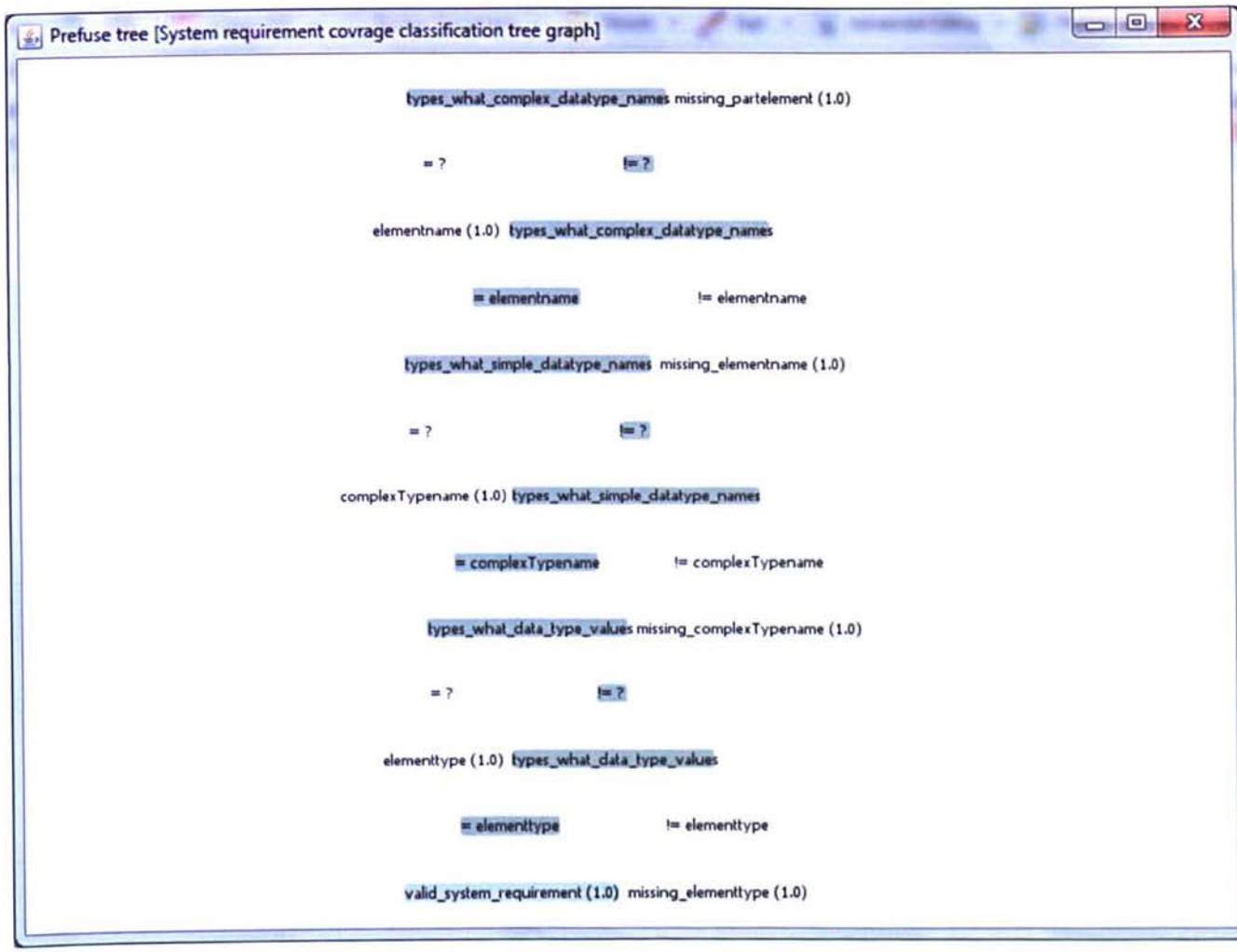

Figure 5.2. A highlighted section of the decision tree structure of the training dataset evaluation of the WSDL Implementation Sztatic classifier

Similarly, the QoS System Requirement-Coverage classifier generates QoS requirement-coverage analysis metrics as structural and machine-learned coverage criteria of the web services under test, and according to the non-functional—QoS standards and specifications within the web services protocol stack. The QoS System Requirement-Coverage classifier uses the outcome of data mining process by the WSDL Static classifier which is saved in WSDL_TO_WSPS table in the metadata database, the table contains the mapping references of the QoS for web services under test specifications to the QoS protocols and standards within the web services protocols stack. This includes messaging,

addressing, security, reliability, transactions, and so on. These protocols and standards support simple and complex QoS requirement message and behaviour patterns between web service provider and consumer. For example, if a web service being tested is advertising QoS as WS-Policy assertions then a set of behaviours concerning those properties can be used in conjunction with the web services provider and the consumer messaging communication patterns.

Accordingly, a training dataset is used to set a classification model of the possible QoS classification for each specified QoS protocol and standard within the web services protocols stack. Each class then set to tuple—sample data patterns which are assumed to belong to each QoS class mapped to syntaxes, semantic, and rule are provided in the training dataset. A predictive machine-learning algorithm—the J48 decision-tree classifier is used to learn the training dataset, modelling a classifier model for the QoS System Requirement-Coverage classifier.

Eventually, the QoS System Requirement-Coverage classifier searches for syntaxes of the QoS behaviour properties and values from the WSDL_TO_WSPS table for the possible QoS data mining. The process of the data mining determines the setting of the web services client's test-harness environment that will be needed in the test case generation and test-execution of each operation.

An example of data mining by QoS System Requirement-Coverage is the knowledge discovery of the QoS semantics, such as the WS-Policy assertions mapped as QoS protocols and standards in the web services protocol stack, from which new QoS requirement-coverage classification cases may be predicted. A decision tree structure of training datasets is built-up for further data mining of the new classification cases of WS-

Policy assertions. Figure 5.3 shows a highlighted section of the structure of the decision tree from the training QoS assertions dataset based on QoS protocols and standards in the web services protocols stack, which is based on QoS standard and specification requirement-coverage.

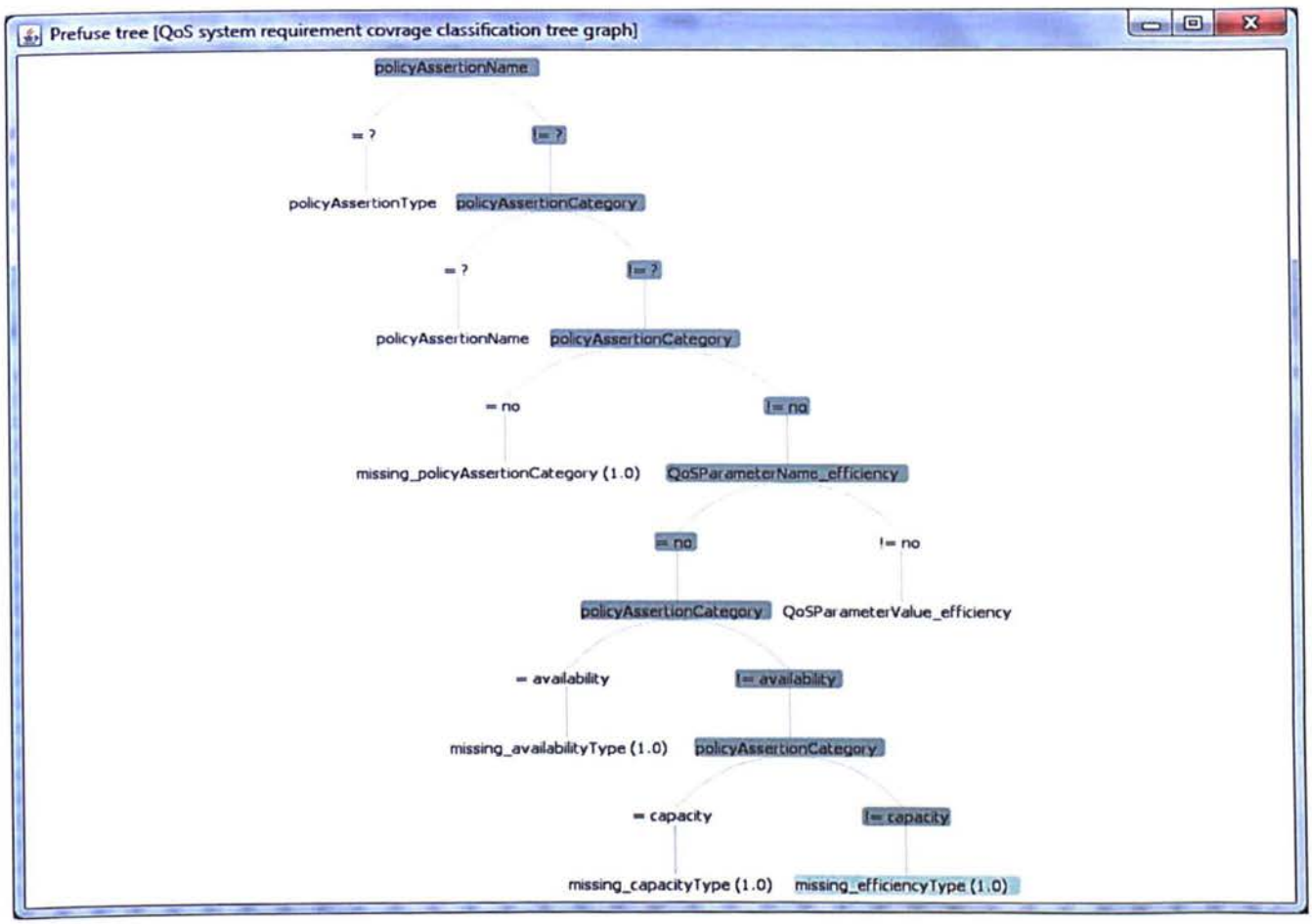

Figure 5.3. A highlighted section of the decision tree structure the training QoS assertions dataset

The QoS data-mining outcome is then used to guide the test-data generation and to derive a test suite including test execution environments.

5.3 Test Generation Example

5.3.1 Test data Category Partitioning

Initially, the test cases generation starts with the CP procedure by capturing and identifying the data from results of data mining of the core and QoS requirements specification of the web services under test, which is saved in the metadata database. The CP procedure decomposes the data into categories of operations, input-output parameter properties, and core and QoS characteristic values. These categories are then identified into partitions of choices of parameter types (a choice is a specific test input and output parameter type or value for a category). For example, choices for primitive parameter types can be selected from primitive data types of programming languages, for example, integer, character, and string. The example below demonstrates the steps of the approach.

Example: web services operation; find city temperature using Java primitive data type as input-output parameters:

1. Classify the web services operations into categories using the CP technique, as shown in figure 5.4:

Category operation find_city_temp	Category operation find_city_temp	Category Core/QoS standard
Input: city_name	Output: city_temp	<wsdl:*>

Figure 5.4. The result set of classifying the web services operations into categories

2. Partition the categories into choices, as shown in Figure 5.5:

Input: city_name Type: string Length: maxLength minLength	Output: city_temp Type: integer Length: integer	<wsdl:*> <wsdl:service> <wsdl :binding> <wsdl:message> <wsdl:types>

Figure 5.5. The result set of partitioning the categories into choices

5.3.2 Equivalence Class Partitioning

The Test Cases Generation module uses ECP procedures to produce equivalence class partitions for partitions of choices for input-output test data. In essence, most programming languages provide values for primitive types which can be mapped to represent most of the input-output XML data type conditions values (constraints). Table 5.2 contains examples of mapping input-output facet XML data types to the constant value of the Java primitive data types [24].

XML data type	XML facets (constrain)	Java constant value
String	enumeration length maxLength minLength whitespace pattern	Integer.MAX_VALUE Integer.MIN_VALUE
Double	enumeration minInclusive minExclusive maxInclusive maxExclusive whitespace pattern	Double.MAX_VALUE Double.MIN_VALUE

Table 5.2. Examples of mapping xml data types to Java primitive data type

(Note: The String class in Java programming language keeps track of the number of characters in the array within integer data type range [101]). Taking that into consideration, the input parameters and QoS conditions values (constraints) can be combined as partitions of data as training datasets for machine-learning classifications. Once the training dataset is available, a machine-learning algorithm is used to learn the classification's rules that relate to generating data sets of category, choice, and constraints-classifications-rules. The Test Case Generation module uses the Naive Bayes classifier to produce equivalence class partitions from the test data which is produced from CP step , which consists of a set of operation names, data types of input-output,

constraints (conditions) which are determined among the choices of CP, the example below demonstrates the steps of the ECP approach:

1. Determine the constraints among the choices using the ECP technique according to the following rules [40]: (a) if the input condition specifies a range of values, then define at minimum one valid and two invalid equivalence classes, (b) if the input condition requires a specific value, then define one valid and one invalid equivalence class, (c) if the input condition specifies a member of a set, then define one valid and one invalid equivalence class. We then include any applicable core functional and QoS standard specification, by implementing CP techniques. Table 5.3 demonstrates the result set of the ECP step.

Input:city_name	Output:city_temp	<wsdl:*>
string>integer.MAX_VALUE,invalid	Invalid error output	<wsdl:service> wsdl:service:name wsdl:address wsdl:port:binding:name
intger.MIN_VALUE=>string>=integer.MAX_VALUE,valid	Valid output	<wsdl:service/> <wsdl :binding> wsdl:binding:name wsdl:binding:transport wsdl:operation:name
string<integer.MIN_VALUE, invalid	Invalid error output	<wsdl :binding/> <wsdl:message> wsdl:operation:name wsdl:input wsdl:output <wsdl:message/> <wsdl:types> Element:name:type <wsdl:types/>

Table 5.3. The result set of the ECP step

2. With the test case generation approach identified, develop a list of test cases frames based on possible inputs and expected outcome for the find city temperature web services method using Java primitive data type as test inputs. The ECP step generates test input values using Strong Normal Equivalence Class Testing methods [40], which uses Cartesian Products for possible input values for each of the input parameters. In Cartesian products, every unit of a group is paired with every unit of every other group. Thus, all combinations of the inputs across all groups are obtained. Hence, the ECP method promotes a design test

which ensures completeness and non-redundancy of test case generation. For the find city temperature method which has one input parameter with three input values, taking into consideration the special case conditions of the input values of the parameter, e.g., lower boundary, upper boundary and zero inputs for each web service method, 6 (6^n) test cases are generated. Table 5.4 shows the 6 test cases which are identified.

TC No	Test case feature	TC validity
1	Consisting of one test case with test input greater than the upper bound	Invalid
2	Consisting of one test case with test input below lowerbound	Invalid
3	Consisting of one test case with test input for correct number of input value	Valid
4	Consisting of one test case with test input of value of the upper boundary	Valid
5	Consisting of one test case with test input of value of the lower boundary	Valid
6	Consisting of one test case with test input of value of zero	Valid

Table 5.4. The list of test cases for the find city temperature web services method

3. The Test Case Generation module then writes XML abstract frames of the test core and QoS test cases, including the test data result from CP and ECP processes. These XML abstract frames also include the web services name, the target name space, the end-point URL, the port name, the operation to be tested, the output method name, the output expected data type, the output element type, the test validity, the test type, and any applicable test environment and QoS condition values.
4. Once the abstract test case frames have been defined, the Test Case Generation module will automatically transform test case frames into XML test-execution scenarios, adding for each test case the actual input test values, the expected result, the test ID, and the test cycle number.

5.4 Test Execution Example

As mentioned earlier, the LDM module also searches for any applicable QoS behaviours mapping the data to WS-* Architecture. This data is also included in the test cases as part of the headers, which are afterward used by the Test-Execution module for test-harness implementation which include systematic automated test environment setup, test-execution, and test-execution response reports of the web services test client. For example, if a web service being tested is advertising WS-Addressing protocol assertions, then the QoS web services rules in the SOAP header description according to WS-Addressing specification are applied. The WS-Addressing specification outlines a set of End Point References (EPR) and Message Addressing Properties (MAP), as well as a set of behaviours concerning those properties which can be used in conjunction with web service provider and consumer communications to support simple and complex message patterns in both asynchronous and synchronous communication types [105]. Table 5.5 list WS-Addressing protocol assertions and applicable mapping web services, as well as consumers' communications rules.

WS-Addressing specification	Expected wsdl infoset	Service type	Description
WS-Addressing disabled	None Input wsam:Action expected	None Addressable service	WS-Addressing headers must not be used by the sender or receiver
WS-Addressing enabled	wsam:Addressing	Addressable service	The service capable of accepting connections on a network endpoint: the wsa:ReplyTo refers to an addressable endpoint
WS-Addressing required	wsp:Optional	Service requires WS-Addressing	wsa:ReplyTo refers to an addressable endpoint is required and wsa:Action property is required by the receiver
Synchronous required	AnonymousResponse	Synchronous service	The service required WS-Addressing and required the use of anonymous response EPRs
Asynchronous required	NonAnonymousResponse	Asynchronous service	Service requires WS-Addressing and requires the use of nonanonymous response EPRs
Request-Response message	Input wsam:Action + output wsam:Action	two-way messaging service	Service is expected to carry on Request and Response messages and requires the use of Request and Response messages MAPs
One-way message	Input wsam:Action only	One-way messaging service	Service is not expected to send SOAP Response and requires the use of Request messages MAPs

Table 5.5. WS-Addressing protocol assertions communications rules mapping

According to the data and mapping rules, the LDM module determines the setting of the web service client's test-harness environment that will be needed in the test-execution of each operation. A dataset can be used as a training set, according to the QoS standards and protocols advertised by the web services under test. A predictive machine-learning algorithm, the J48 decision-tree classifier [61], is used to learn rule pairs (category, choice) that relate to the web services, modelling input properties to output-domain equivalence classes according to the discovered WS-Addressing XML Infoset in the web services under test WSDL. Figure 5.6 shows a section of the decision tree structure from the training QoS WS-Addressing dataset. The LDM module searches for QoS Infoset values in a specific table in the web services specification metadata database, from which new QoS classification cases may be predicted, mapping the QoS implementation of web services to communication types between the test consumers of the web services under test. The Test-Execution module uses these settings to implement the test harness of the web services test client at the time of creation of the SOAP message asynchronous or synchronous communication dispatcher.

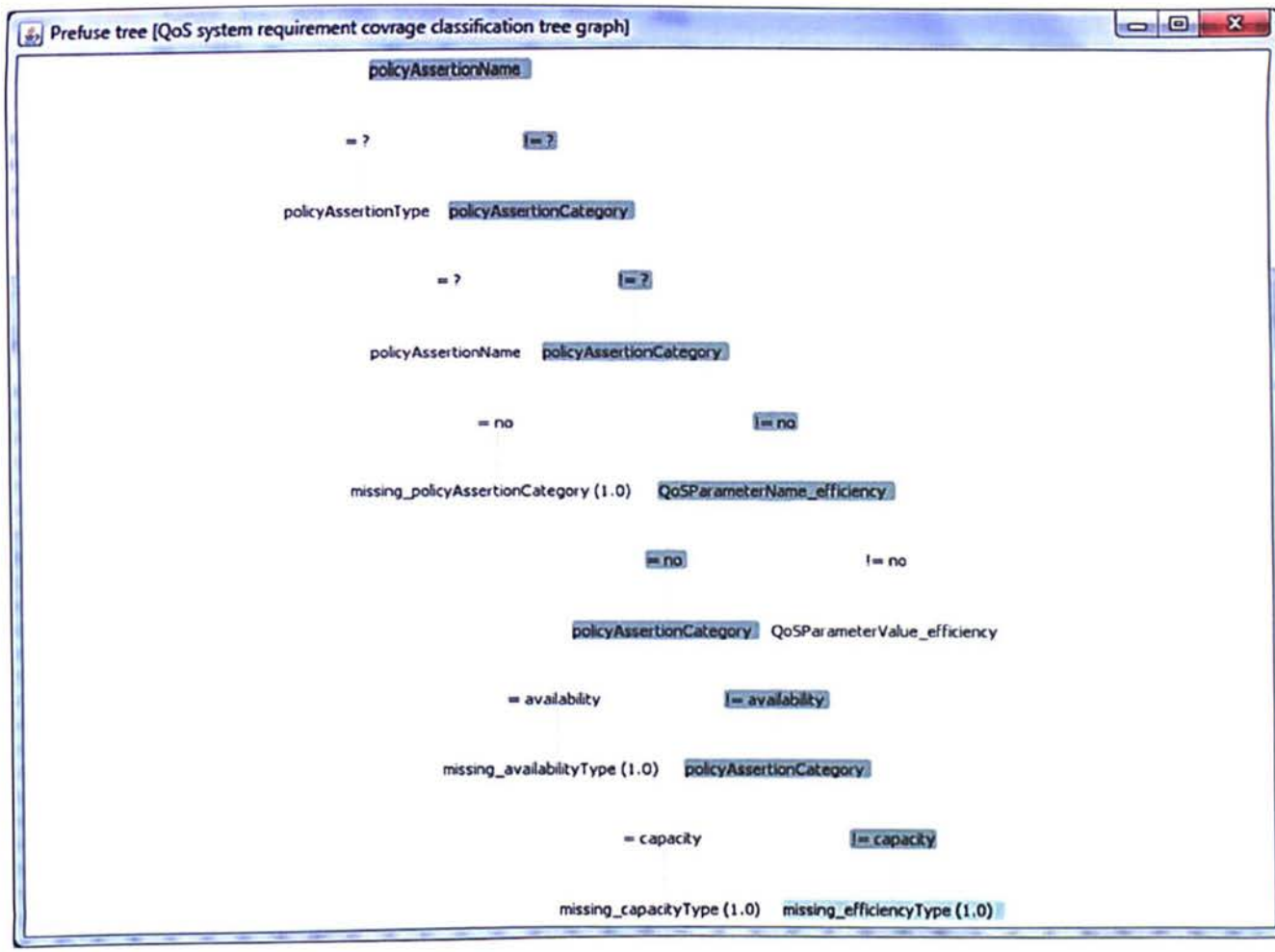

Figure 5.6. A section of the structure of the decision tree from training QoS WS-Addressing dataset

A web services prototype is developed as a proof of concept to test the invocation of web service operations according to the test execution environment of the QoS WS-Addressing protocol. The web services under test provide live stock information through web service operations: ticker last price, ticker price change, and last change date. The web services implement WS-Addressing, and in accordance with that, the WS-Addressing XML Infoset appears in the WSDL. Figure 5.7 shows the applicable WS-Addressing protocol properties in StockTickerPrice.wsdl.

Figure 5.7. WS-Addressing protocol properties in StockTickerPrice.wsdl file

The LDM module makes a decision based on training datasets, by capturing the specific QoS WS-Addressing protocol data retrieved from the web services specification metadata database, and on the basis of this the test-harness environment is predicted. The figure shows that the test-harness environment data is included in the test cases as part of the headers. Figure 5.8 specifies the test execution environment header in the Test Case.xml file.

Figure 5.8. Specifying test-execution environment data in the Test Case.xml

Supporting both polling and call-back mechanisms when calling web services asynchronously, the Test-Execution module and Monitoring module implement the test-harness for web services under test, sets the input parameters in the SOAP Body, and executes the test case systematically.

5.5 Test Monitoring Example

The Monitoring module performs run-time operational monitoring of the web services log file for messages and logs sent between the web services test client and web services in real-time. The Monitoring module generates monitoring statistical reports, the Test-Execution module initiates the dynamic testing process by executing the XML test-execution files using the unique test ID to identify, match, and update the test execution result for each test execution simulation task. Simultaneously, the Test-Execution module waits for the response messages sent from the web services under test in real-time. The Test-Execution module generates a test execution report, by transforming the SOAP

message to string of data for parsing and extracting the test response data. The response data includes the Test ID, the response Body, and the message content result. The Test-Execution module updates the test_execution_data table in the test database with the test execution result for each case according to its unique test ID number. The test simulation outputs at the end of each test cycle are used within the monitoring process with support of LDM module for test validation in the next stage by Monitoring module. The Monitoring module then reports these messages and transforms them to sort of customised message data which can be used by the LDM module and then saves this data in the System_Historical_Data database. This data is considered the final observational learning inputs throughout the monitoring process. Eventually, the LDM module analyses and judges whether the components of the web services under test have met the requirements and the system operational standards, otherwise it needs to expose new test experiences and proper debugging efforts to be directed. The results of the testing data metrics become an input for the trained machine-learning classifier. Using the J48 decision-tree classifier, a new classifier module is trained with the training dataset to validate against the testing dataset. Figure 5.9 shows a section of structure of the decision tree from the system invocation monitoring training dataset.

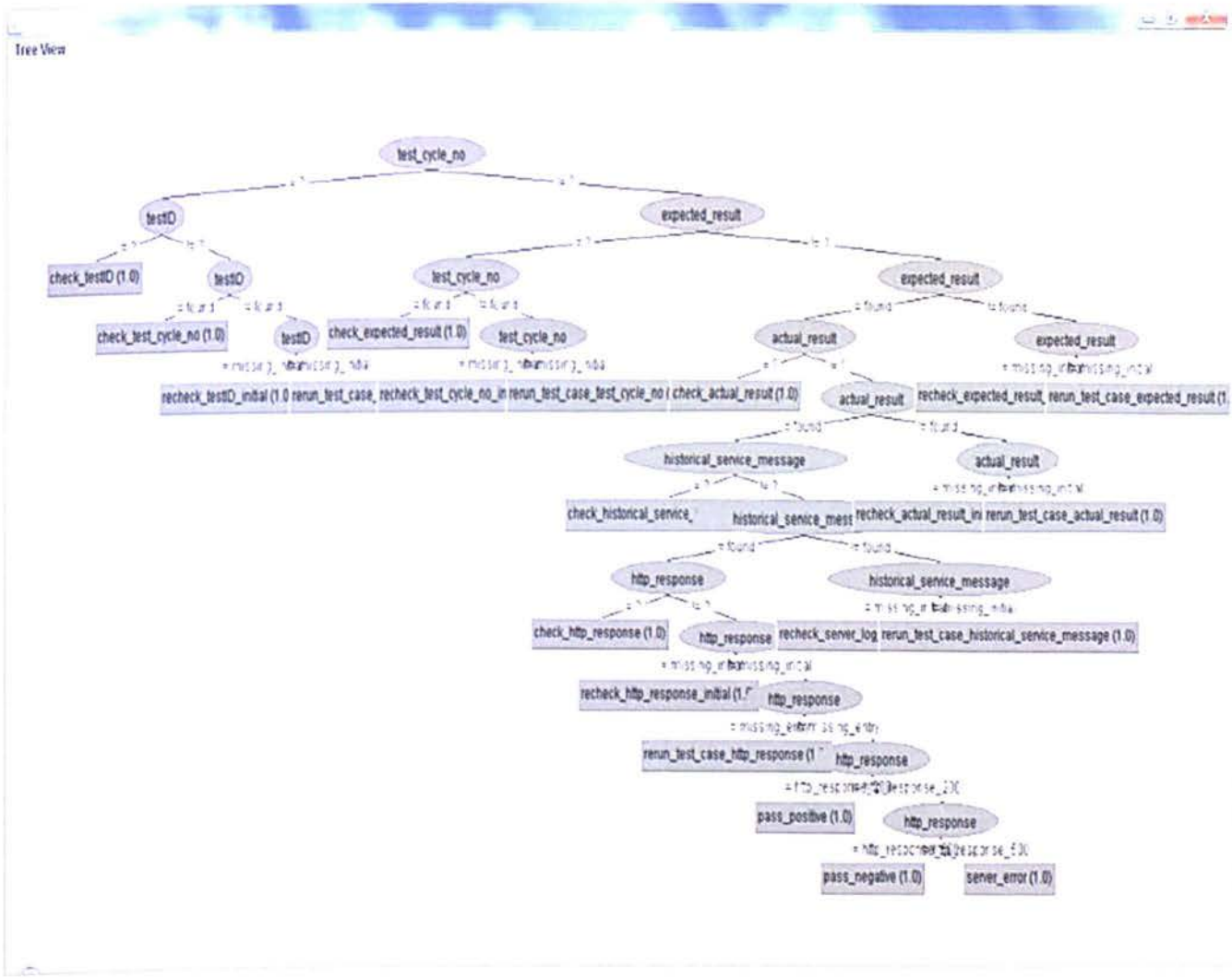

Figure 5.9. A section of the structure of the decision-tree resulting from system invocation monitoring training dataset

The LDM module analyses the test results in the historical database in order to judge whether the test executions of the test cases meet the expected results and requirements of the QoS system's operational standards. The LDM module then feeds the analysis of the results to the test database.

5.6 Empirical Framework Evaluation

As mentioned in the introduction section of this chapter, the cost-effectiveness matrices deal with capturing, defining, monitoring, and analysing software testing techniques and

practices exclusively by using certain key factors within the testing phase [78]. These factors are considered essential for producing concrete structures of software test techniques which could prescribe their quality levels in a cost-effective manner. These key factors are as follows [16],[79],[70]:

1. The total effort spent on the testing phase (test cost).
2. The total defects captured at test phase (defect detection effectiveness).
3. The defect detection effectiveness which is computed against test suite size (test cost) to obtain the cost-effectiveness or test-efficiency ratio (defect detection rate/test cost).

Since all the testing technique structures are, to various extents, based on heuristics and simplifying assumptions [16],[79], the cost-effectiveness of various techniques cannot be systemically assessed and compared. Hence, it is natural to utilize empirical analysis in order to compare and improve software testing techniques and practices. Thus, in the following sections and subsections, the proposed framework will be empirically evaluated comparing it against other frameworks and tools

5.7 Defect Detection and Coverage Metrics

5.7.1 Test Completeness Measurement

In order to demonstrate that the test suite which was generated by the proposed framework satisfies the coverage criteria according to the requirements of the web services under test, test cases were generated using a white-box (unit testing) testing tool. The JUnit tool [54] was employed to generate and execute unit tests for the same source code, of the web services under test, used by the proposed framework for generating test cases by the Test Cases Generation module. The generated Unit Test

Cases (UTC) include the same combination of input test data, method calls, and expected outputs which are used during the test-case generation through the proposed framework. Then, the CodeCover tool [23] (an open source instrumentation tool which is integrated into JUnit) was used to independently gauge the coverage level achieved by a test suite through instrumenting the source code. Thus, the code coverage metrics can be generated for each unit test case providing the percentage of code that is covered by the unit test case, i.e. the proportion of requirements that have been satisfied [16],[26]. The CodeCover tool supports statement (instruction) coverage, branch coverage, line coverage, complexity coverage, and method coverage. For implementing the test coverage measurement, the calculator web services test suite generated by the proposed framework is employed. The test suite is composed of test cases for testing the web service methods of the calculator web services, with two input parameters of Integer type and one output of Long type for each method. Having two input parameters, and six different values for each parameter for the four methods (Add, Subtract, Multiply, Divide) of the calculator web services. Using the Cartesian product (as exemplified in section 5.3.2), every test input of a unit test case is paired with every test input of every other unit test case. Appropriately, all combinations of the test inputs across all unit tests are obtained, 144 (6^2)*4 of test cases are identified and generated. Then, the JUnit tool was employed to execute the test suite. Accordingly, by using the CodeCover tool to generate the coverage rate percentages covered by each unit test; various coverage percentage values are generated for 144 unit test cases for the calculator web services. The graph in Figure 5.10 summarizes the results of the source code instrumentation.

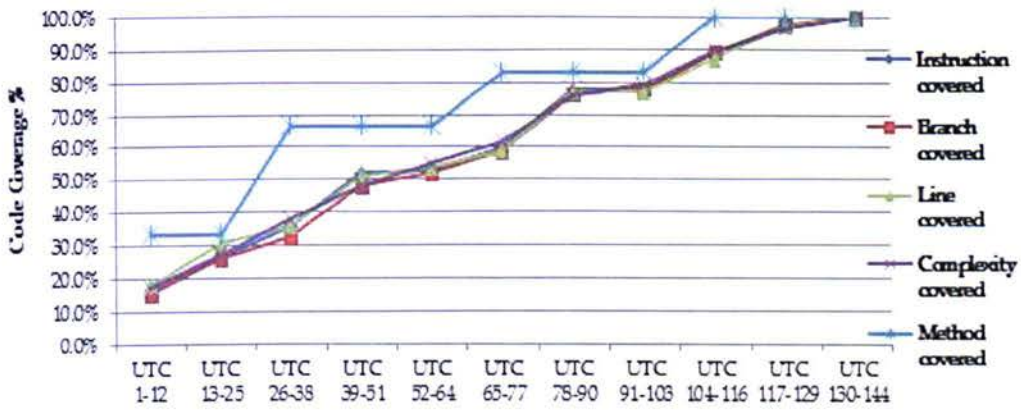

Figure 5.10. Code coverage measurement for calculator web services

As the unit test case is iterated (1-144), the code coverage was found to be improving, reaching 100% coverage, for all, i.e. for instruction, branch, line, complexity, and methods coverage percentage values. Hence, this result can attain *full coverage criteria* according to web services under test requirement.

5.7.2 Defect Detection Effectiveness Measurement

In order to measure the second factor of cost-effectiveness -the defect detection effectiveness- the mutation score [122] can be used to measure the effectiveness of a test set in terms of its ability to detect defects, which gives an indication of the defect detection effectiveness of the test suite.

5.7.2.1 Defect Seeding

Mutation testing is performed by selecting a set of mutation operators and then applying them as manually seeded defects (or mutants) to the source code of the web services under test, one at a time for each applicable piece or block of the source code. The outputs from the running test suite are then compared against the web services under test. If the test suite is able to detect the change, (i.e. one of the tests fails its positive testing) then the mutant is said to have been killed (detected) and the test suite is

successful. When all mutants have been killed, the saved test case is comprised of the test suite, which could be used to test the web services. There are three kinds of mutation operators available, namely statement-level operators, method-level operators and class-level operators [77]. For our mutation testing, the statement level operators are chosen for measuring the mutation score, because this involves the creation of a traditional set of code-line-level mutants for the web services being tested. Mutation operators for method and class-level mutation testing focus instead on testing object-oriented specific features, for example inheritance, polymorphism, dynamic binding, and encapsulation. Moreover, the seeded defects in our mutation testing should include an adequate number of mutations (or defect) types that cover the unit test cases, based on the "selective mutation" operator set introduced by Mothra [25]. Table 5.6 shows seven mutation types were chosen for selective mutation operator set.

Operators	Description	Example
AOR	Arithmetic Operator Replacement	Modify arithmetic operator (+, -, *, /, =, ++, -, +=, -=, *=, /=)
LCR	Logical Connector Replacement	Modify logical operator (&&, ||)
ROR	Relational Operator Replacement	Modify relational operator (>, <, >=, <=, ==, !=)
SAN	Statement Analysis	Invert the condition statement.
RSR	Return Statement Replacement	Set / return different variable attributes
SVR	Scalar Variable Replacement	Set / return different integer value
UOI	Unary Operator Insertion	Set / return different Boolean value (true, false)

Table 5.6. Selective mutation types from Mothra Mutant Operators

The seven mutation types that we have been selected and then applied as 18 manual seeded defects, some of different types while others of the same type, to the source code of the calculator web services one at a time. After running the unit test suite, the killed mutants (detected defects) metric of the test suite was constructed, and the defect detection rate of the test set was measured according to the mutation score. The mutation score takes real values between 0.0 and 1.0, where 1.0 is the best score

possible, meaning that this particular test set can kill all the non-equivalent mutants (there are some mutants that can never be killed because they always produce the same output as the original program, these mutants are called (Equivalent Mutants). Such a test set is said to be 100% mutation adequate to measure the rate of defect detection of the test suite [56]. The graph in the Figure 5.11 summarizes the mutation-detection effectiveness of the generated test suite by the proposed framework for the (Add) method of the calculator web services. The graph illustrates that the test approach adapted by the proposed framework is capable of finding all defects of all the selective mutation types with 100% mutation score. The horizontal axis signifies the mutants, while the vertical axis signifies the percentage of the test cases which detected the mutants, for example, defect 1 was found by 19.4% of the test cases while defect 10 was found by 36.10%, still all defects were found. The area under the curve represents the mutation-detection ratio, which shows the detected defects over the life of the test suite.

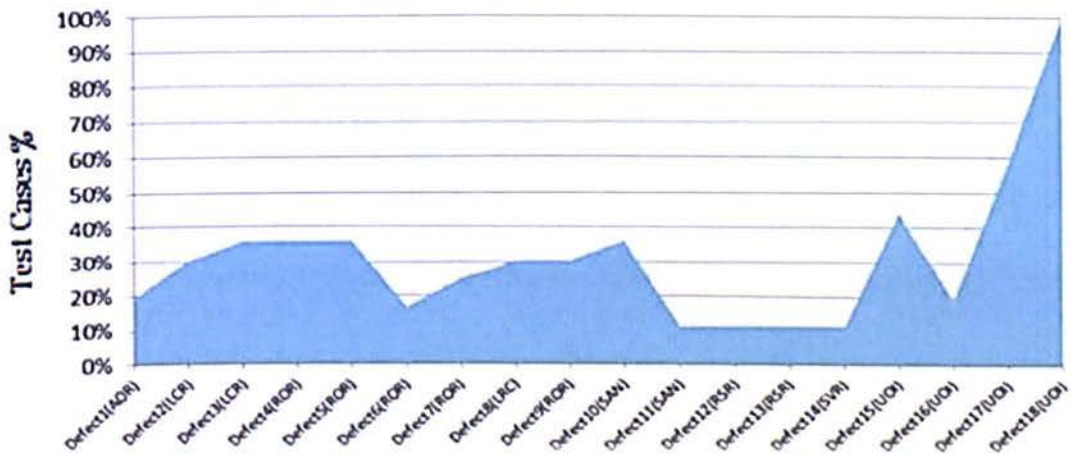

Figure 5.11. Defect detection ratio for calculator web services test suite

Table 5.7 shows mutation testing metrics for the (Add) method of the calculator web services corresponding to the graph - Figure 5.11.

Defect Types	Unit Test Cases (Test Cost Size)																																	UTC Coverage %	Detected Defects
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36		
Defect1 AOR															x	x	x	x			x	x							x								19.40%	7
Defect2 LCR			x	x											x	x	x	x	x		x	x		x					x								30.50%	18
Defect3 LCR			x	x							x				x	x	x	x	x		x	x		x		x			x								36.10%	31
Defect4 ROR	x			x							x				x	x	x	x	x		x	x		x		x			x								36.10%	44
Defect5 ROR	x			x				x							x	x	x	x	x		x	x		x		x			x								36.10%	57
Defect6 ROR	x			x											x				x		x			x													16.60%	63
Defect7 ROR	x			x											x				x		x			x	x			x			x						25.00%	72
Defect8 LRC	x			x											x				x	x	x		x	x	x			x			x						30.50%	83
Defect9 ROR	x			x											x				x	x	x		x	x	x			x			x						30.50%	94
Defect10 SAN			x		x	x		x				x		x		x	x	x				x					x			x		x					36.10%	107
Defect11 SAN									x		x	x	x																								11.10%	111
Defect12 RSR									x		x	x	x																								11.10%	115
Defect13 RSR									x		x	x	x																								11.10%	119
Defect14 SVR									x		x	x	x																								11.10%	123
Defect15 UOI	x			x			x				x	x	x		x				x	x	x		x	x	x			x			x			x			44.40%	139
Defect16 UOI	x			x			x			x	x	x	x		x				x	x	x		x	x	x			x			x			x			19.40%	156
Defect17 UOI		x	x		x	x		x	x		x	x	x	x		x	x	x				x				x		x	x		x	x		x	x		58.30%	177
Defect18 UOI	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	100.00%	213
	9	11	14	27	30	33	36	40	43	46	50	53	56	59	63	66	69	72	76	79	82	85	88	92	95	98	101	105	108	111	114	117	120	123	126	129		

Table 5.7. Defect detection metrics for the generated mutation testing

The results in table 5.7 indicate that mutation-detection effectiveness is ranges between 19 and 100% presented in UTC Coverage % column. The Defect Types column signifies the mutants, while the numbers are shown in Detected Defects column which signifies the number of detected defects.

5.8 Cost-effectiveness Measurements

In order to measure the cost-effectiveness of the test approach which is adapted by the proposed framework, the defect detection effectiveness was calculated and plotted against the test suite size (test cost) [16]. The resulting graph is presented in Figure 5.12, where the horizontal axis represents the test suite size or test cost, while the vertical axes represent the detected defect rates (UTC Coverage % column—Table 5.7). The figure shows the cost-effectiveness line which illustrates as the test suite size increases (first row in green—Table 5.7), the number of detected defects increase (last row in red- Table 5.7), which provides evidence of the defect detection adequacy criteria. The graph also

illustrates the defect detection effectiveness line of the test execution cycles, as more mutants (defects) of different mutation types are found when new mutants are injected with zero undetected defects. This provides in-process evaluation of the actual test suite effectiveness at detecting defects [74]. Furthermore, the figure shows the confidence proportion of defect detection around the cost-effectiveness line, which provides a further insight into the dependence of defect detection probability on the test suite coverage, as very often, many adequate test suites can be found for a given coverage criterion [16]. The fit of the defect detection to the test-case coverage demonstrates that the proposed framework is consistent which indicates high cost-effectiveness from the perspective of the test coverage.

Many of the cost-effectiveness parameters and measures depend on other validity factors that were not addressed in this empirical analysis, and which could reflect on the confidence given to the study outcomes. These factors ensure that any cost-effectiveness comparison among test techniques is unbiased. The following subsections discuss the different types of threats to the validity of evaluating the cost-effectiveness of the proposed framework.

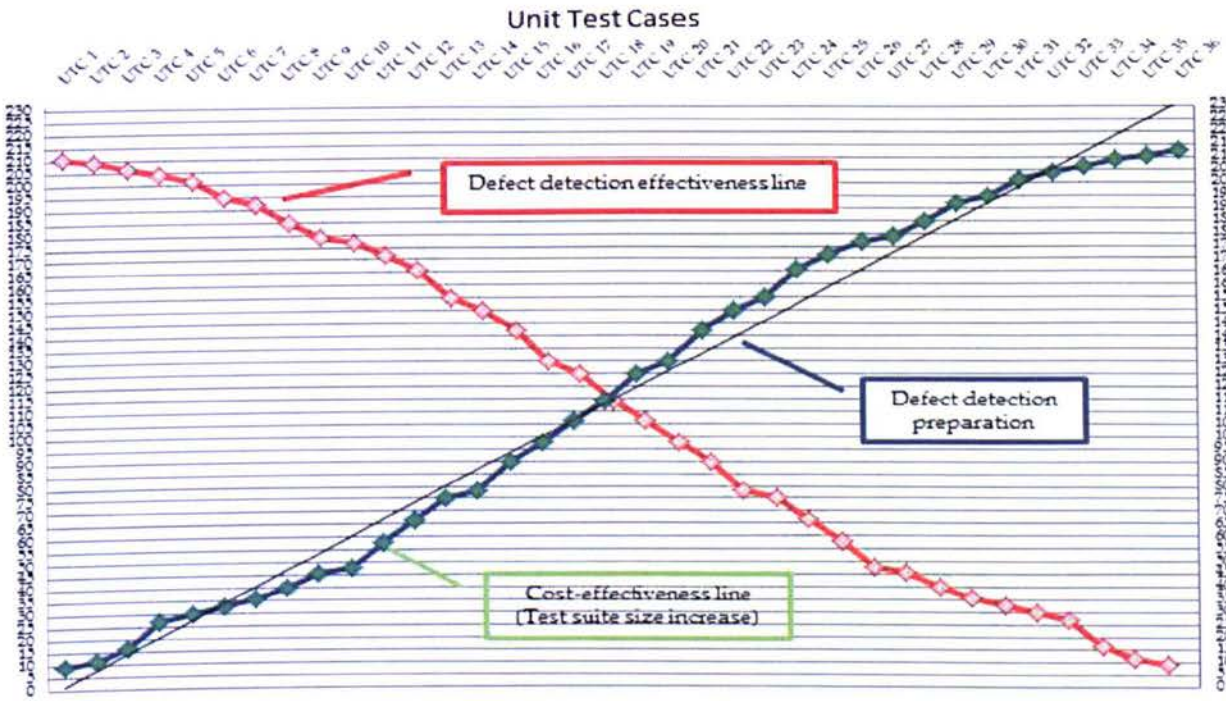

Figure 5.12. Analysis of test cost-effectiveness

5.9 Threat to Construct Validity

One threat applicable to construct validity is related to the effort required or the test cost when the required resource is machine or human time for test-case generation, test-execution, and collecting and logging the test result, the test cost can be measured in terms of actual time is needed to generate and execute a test case. In order to address this validity issue, the overall test cost is calculated, the proposed framework was used and compared against other open-source and commercial tools for SOA testing. The test cost is estimated by measuring the actual time required to generate and execute a test case along with the steps presented by the proposed framework, they were then manually compared against generated ones, by mimicking a human tester using the other benchmark open source and commercial tools for testing SOA for SOA testing. In particular, this evaluation, one test case is used for testing the addition method (add) of

the calculator web services and the result of the comparison is collected for the *generation and execution test case* with a test input of value of the upper boundary. Figures 5.13, 5.14, and 5.15 show the result of the comparison of the test task expended by the proposed framework for generating and executing a test case for the addition method (add) of the calculator web services with four leading and comprehensive SOA test tools, namely HP Service Test [46], SoapUI, Soapsonar, and Parasoft SOAtest. In the graphs, the vertical axis shows the SOA test tool or framework, while the horizontal axis shows the work-time consumed (total actual time in seconds) in the test-case generation and in the test-harness implementation which includes test environment setup, test-execution, and test-execution response of each test case by the SOA test tools. The results demonstrated reduced testing effort (test cost) using the proposed framework as compared with other SOA test tools. The actual results show that the proposed framework clearly required the lowest amount of time of these SOA test tools—7.9 seconds for generating and executing a test case for a single web services method per test cycle. Other SOA test tools required significantly more time than the proposed framework—on average 47.5 seconds for generating and executing a test case for a single web services method per test cycle.

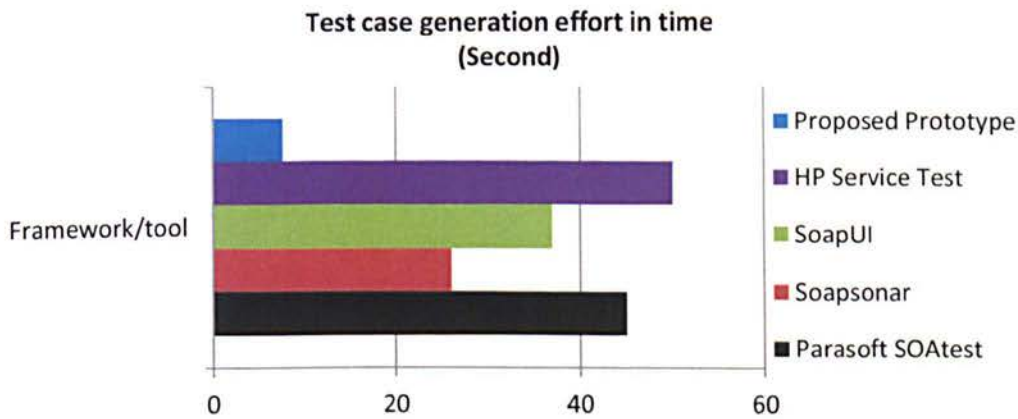

Figure 5.13. A comparison of test-case generation cost

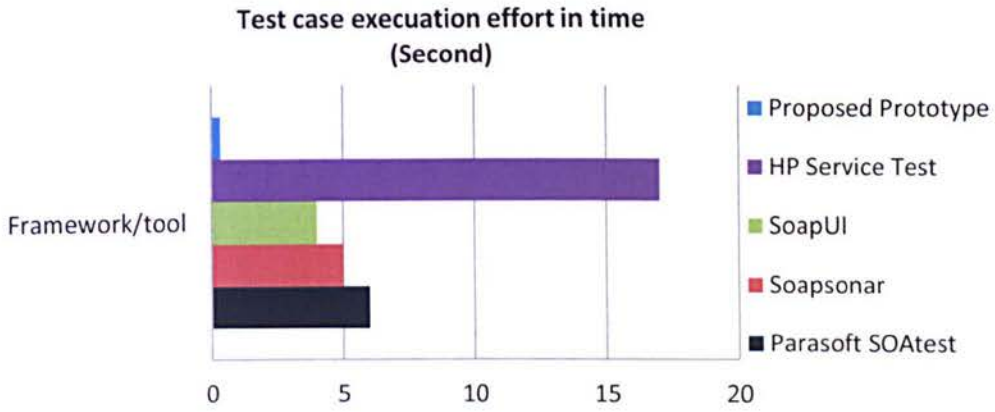

Figure 5.14. A comparison of test-case execution cost

Figure 5.15. A comparison of test-case execution response time

Computing infrastructure: the evaluation is conducted on two computers with Processor Intel(R) Core(TM) i3 CPU M 370 @ 2.40GHz, 2399 Mhz, 2 Core(s), 4 Logical Processor(s) at 3.2GHz, 4GB of RAM, running Windows 7, XP.

5.9.1 Threats to Internal Validity (Degree Level of Automation)

A relevant internal validity factor is related to the degree level of automation of the proposed framework for testing SOA. This validity issue could be determined by comparing it against other open source and commercial tools, according to test automation validity factors [16]. Figure 5.16 shows a comparison graph which tracks the

degree level of test automation metrics of the proposed framework and other benchmark *open source and commercial tools for SOA testing.*

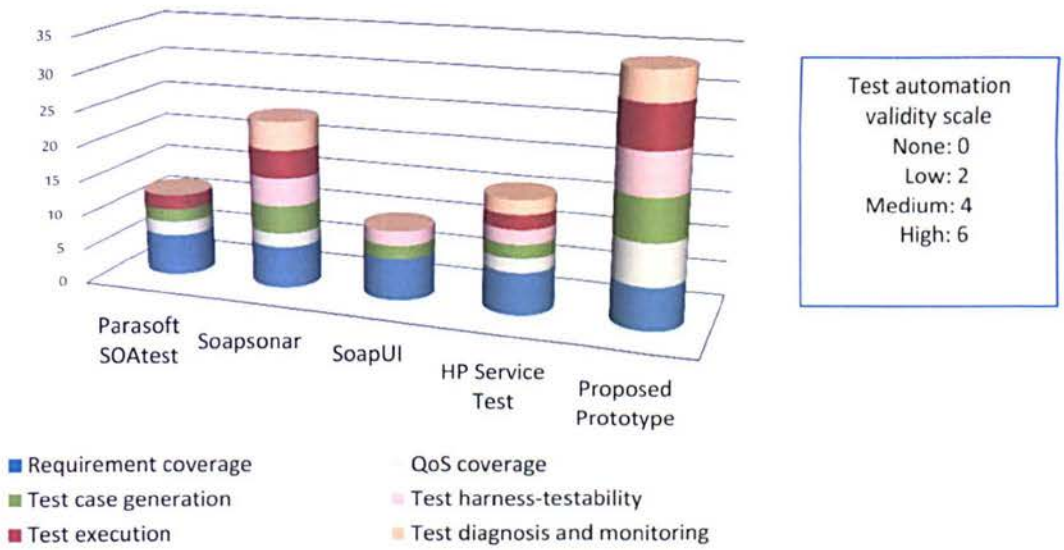

Figure 5.16. A comparison of level of automation of the proposed framework against other benchmark open-source and commercial tools

5.9.2 Threat to External Validity (Supporting Industrial Practices)

There are several external validity threats which limit the ability to generalize the design and implementation of the proposed framework according to industrial practices for SOA testing, e.g. WS-* Architecture implementation [51], [55]. These external validity issues can be resolved by properly addressing and implementing them in the proposed framework, and also using them as comparison factors for cost-effectiveness measurement against similar frameworks and techniques from the literature. Figure 5.17 shows a comparison graph which covers and captures the level of implementation of these factors in the proposed framework and in other similar frameworks and techniques from the literature.

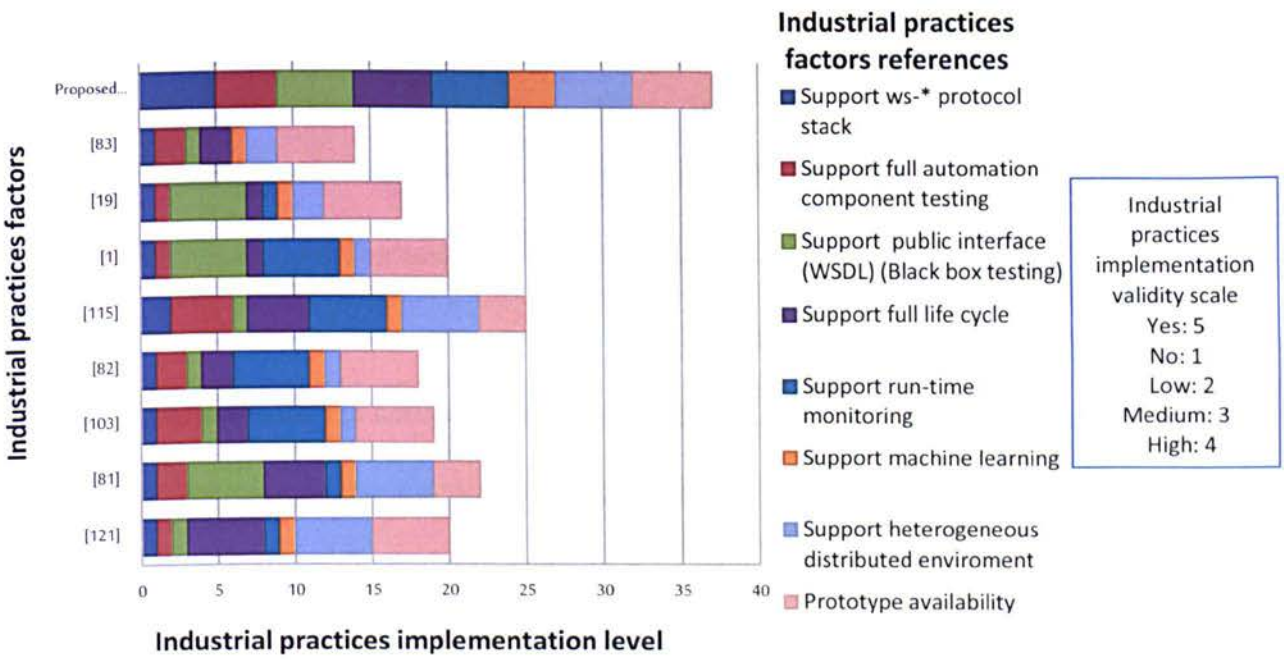

Figure 5.17. A comparison of level of automation of the proposed framework against other benchmark open-source and commercial tools

5.10 Cost-Effectiveness Evaluation Summary

After detailed presentations of individual modules of the framework together with working examples, an empirical analysis was carried out in order to evaluate the cost-effectiveness of the proposed framework by using key factors such as test cost, defect detection effectiveness, and cost-effectiveness measurements. A measure for the first key factor, the test cost, was conducted to evaluate the test suite completeness generated by the proposed framework as a measure for the test cost. The test suite completeness result has shown full code coverage by the test suite which implies that the test suite generated by the proposed framework has a high degree of test cost-effectiveness with requirement specification traceability. An internal validity factor for test cost evaluation has been addressed through a practical comparison of the proposed framework against four leading and comprehensive open source and commercial SOA

test tools: HP Service Test, SoapUI, Soapsonar, and Parasoft SOAtest. The results demonstrated reduced testing effort (test cost) using the proposed framework as compared with other SOA test tools. The actual results show that the proposed framework clearly required the lowest amount of time of these SOA test tools. The degree level of automation, another internal validity issue, was also established and determined through empirical comparison of the proposed framework against the other SOA test tools according to the automation validity factors. The results verified that the proposed framework attains good automation in comparison to other SOA test tools.

An external validity factor which could limit the ability to generalize the proposed framework for supporting SOA industry practices is also determined through an empirical comparison of the level of implementation of these factors in the proposed framework against similar frameworks and techniques from the literature; the results show that the highest level of implementation of these characteristics is in the proposed framework, in comparison to other frameworks from the literature. This verifies that the proposed framework implements and supports the standards and protocols that make it more transparent and applicable in the implementation of SOA systems according to industry practices. Another practical evaluation carried out to measure the second principal key factor, defect detection effectiveness, has shown that the test approach adapted by the proposed framework is efficient and capable of finding all injected mutant types with 100% mutation score, which indicates a high degree of defect detection. A conclusive measure of the cost-effectiveness of the proposed framework was computed and provided evidence of high adequacy test suite for a given coverage criterion, high defect detection adequacy criteria and high degree of cost-effectiveness. All tests based

on core functional and QoS system requirements were executed. The test oracles which are based on data-type references are fully automated.

Chapter 6 – Industrial Case Study

6.1 Introduction

In this chapter, the proposed framework detailed in Chapter 4 and 5 is evaluated through an industrial case study. The chapter presents an evaluation of the effectiveness of the proposed framework by practical and systematic implementation on a generic business use case within different industry sectors. The industry case study is designed and implemented as a prototype system based on a business use case of a SOA web services environment. The environment deployed the majority of the predominant WS-* Architecture specifications such as WS-Transactions, WS-Security, WS-Chorography, WS-BPEL and other standards and specifications. The implemented case study of the SOA web services prototype system is designed to generalize and increase the scale of supporting business activities and the context of usage of standardised industry protocol within the web services protocol layers. The evaluation aims to reach additional conclusions for the potential improvement of SOA testability on each possible industry usage situation.

In this chapter, Section 6.2 revisits and reviews the use of SOA in industry; Section 6.3 describes the setup of the case study. The evaluation of the case study using the proposed framework is conducted in Section 6.4. Finally, we discuss the evaluation results and threats to validity in Section 6.5.

6.2 Scope

Due to the fact that SOA web services are often seen as the foundation of a new generation of Business-to-Business (B2B) and Business-To-Business Integration (B2Bi)

they are considered the key mechanisms for enterprises to gain competitive advantage. Furthermore, web services are becoming truly pervasive by taking the full benefits of the rich capacities of WS-* Architecture such as transaction, security, choreography, and orchestration standard and specifications. Hence, much of SOA implementations are *becoming collaborations of choreography and orchestration of services.* Each web service will provide some large or small functions for the businesses' needs, and the majority of businesses will be able to simply choreograph and orchestrate how these services communicate. Given this wide spectrum of SOA web service implementation within industry sectors, and the relative novelty of the field (i.e. the testing and test analysis of SOA web services), this chapter will attempt to conceive a case study that addresses the different aspects of the field as fully as possible, and according to the literature review on market and industry demand.

An advanced SOA web services system prototype deployment is proposed for integrating security, choreography, and orchestrating standards and specifications in the testing cycle, starting at the choreography level and showing how the requirements map through the different levels of web services protocol stack layer abstractions (see Section 2.3.4.1). Abstract domain-level models are linked to their technical implementation and show how requirements are realized through prototype components in a target architecture based on a web service protocol stack framework. In the following section, the scenario case motivated by testing implementation from security to transaction is implemented, to choreograph and orchestrate. The machine-learning module is deployed together with all the required modules within the framework as proposed in Chapter 4.

6.3 SOA in Industry Segments

Based on existing reports from a number of credible sources ,such as the Gartner reports [37], regarding the popularity of SOA we can conclude that SOA is at the centre of the technologies used to implement e-business. Moving forward, SOA markets are dominated by business process analysis and workflow products. The fact that budgets are in place that pay for existing integration functionality, drives market growth for SOA [87]. SOA automates many business processes in a manner not accomplished by custom coding. Furthermore, SOA offers improved efficiency and significantly lower costs for integrating systems and implementing supply chain efficiencies, thus providing a market driving force. The increased importance of SOA ensures that web services will be widely used, affecting virtually all types of development tools and runtime middleware. Web services are one of important factors affecting middleware markets.

6.3.1 Web Service Protocol Stack Industry Implementation

WS-* Architecture provides end-to-end testing of all three layers: the service layer, service composition and coordination layer, business process and collaboration layer. The major specifications for defining business processes on the business process and collaboration layer, are the WS-BPEL, the WS-CDL , and BPM (Business Process Modeling) built on WSDL which are drawing the most interest from the industry [87]. Business Processes Languages allow for establishing metadata business protocol specifications. This metadata intends to establish a common understanding of the meaning, by specifying syntax and semantics of the data exchanges. These business processes are graphs of activities that carry out meaningful business operations. Examples are purchasing an airline ticket, managing inventory in a warehouse, and ordering furniture

for a home or office. Long-running transactions, such as tracking an order to fulfilment or supporting collaborative planning and forecasting are also business processes.

In order to implement a business process using web service technology the flow of a business process to a set of web service interactions need to be mapped. Services will happen dynamically at runtime, heralding a new era of Business-to-Business integration over the Internet. B2B and B2Bi describe electronic commerce, or e-commerce, transactions between businesses (as opposed to between businesses and consumers). Businesses that engage in electronic commerce transactions are called trading partners, and can include retailers, manufacturers, suppliers, and marketplaces.

6.3.2 Enabling Business Process Layer

Companies implementing business process solutions in the context of scalability and enterprise wide solution sets are achieving significant competitive advantages and improvements in productivity. Areas of demand include financial services, customer relationship management, e-government, and e-business. The WS-* Architecture is seen as an enabling technology for a broader use of web services in B2Bi [111]. Although, the second-generation web services specifications using WS-* Architecture are able to provide comprehensive QoS support, interoperability is still one of the core requirements for B2Bi which use requirements cannot be met by all WS-* Architecture implementations [87].

6.4 SOA Architecture Use Case Implementation

The SOA architecture used a case prototype implementation, consisting of web services of an online stock trading system, as shown in Figure 6.1. The online stock trading web

services system, covering stock-buy-sell business activities, is used in this case study prototype in order to further illustrate and evaluate the framework testing technique. The online stock trading web services system in this practical use case, consists of web services components, each of these web services components implement a web service interface and multiple simulation client web service, residing in different *locations*. *All* web services under group testing are implemented with the same hardware and software specifications. Table 6.1 lists the core functional and QoS system requirements specifications of the online stock trading web services according to WS-* Architecture specifications.

Services	Component type	WS-*architecture specification	Description
Authentication web service	Authentication web service	WSDL WS-ReliableMessaging WS-Security WS-Policy WS-Addressing	Using symmetric encryption based on a single secret key known only to the parties involved in an exchange of messages between a client and a server . Using symmetric encryption, WS-SecureConversation builds on both WS-Security and another standard, WS-Trust. WS-Trust itself builds on WS-Security, defining an interface for a web, service that issues and works with security tokens.
Subscription Account Management web service	WS-Chorography service WS-ReliableMessaging (In principle) service	BPMN2.0 WSDL WS-Chorography WS-Security WS-Policy WS-Addressing	Using choreography service composition, participants Play expected messaging behaviour it in terms of the sequencing and timing of the messages that they can consume and produce as peer-to-peer interactions.
Buy-Sell-Stock web service	Transaction web service	WSDL WS-AtomicTransaction WS-Security WS-Policy WS-Addressing	Using transaction protocol, participants, to coordinate in an interoperable manner between heterogeneous transactions infrastructures.
	Coordinator service	WS-coordination	Manages the transactional state (coordination context) and enables Web services and clients to register as participants.
	Activation service	WS-coordination	Enables activate transactions and create coordination contexts. Once created, the coordination contexts are passed to the transaction service.
	Registration service	WS-coordination	Enables an application to register as a participant.
User Account Management web service	PBEL service	PBEL WSDL WS-Security WS-Policy WS-Addressing	Using orchestration web service, which specifies an executable process that involves message exchanges in interoperable manner between heterogeneous infrastructures.

Table 6.1. Online stock trading web services system components according to WS-* Architecture

The proposed practical use case is a web services Business-to-Business (B2B) one covering secure transaction characteristics with business process solutions based on SOA. The idea is to define adequacy criteria to test SOA web services. This specification has value in state-full secure e-commerce systems, especially in the B2B world, where services are

invoked dynamically according to the result of the previous invocation. The web services offer online stock trading web services and client web services that can access these functions individually. The online stock trading web services system-testing environment shows in Figure 6.1.

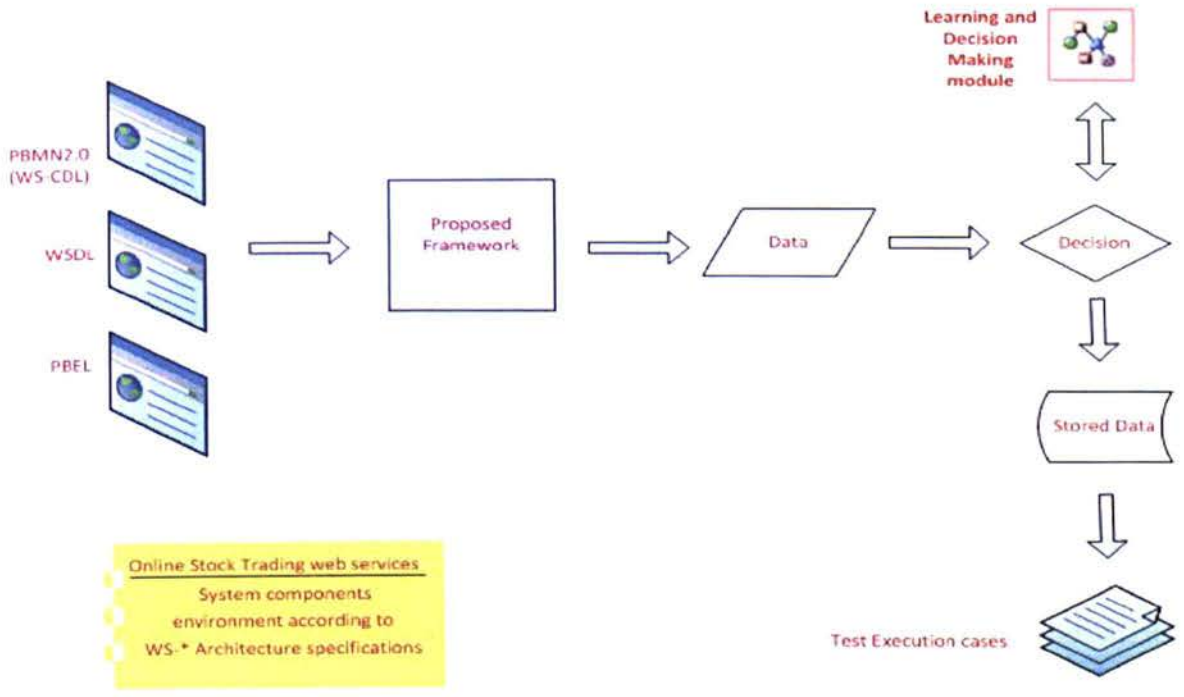

Figure 6.1. Online stock trading web services system-testing environment

6.4.1 Online Stock Trading Process

In a typical client's stock trading requests from online stock trading web services, the flow of information is as follows:

1. Login: clients communicate with the authentication web service using reliable messaging (WS ReliableMessaging) with security mechanisms (WS-Security), and using addressing location (WS-Addressing) within collection of QoS policy (WS-Policy) protocols, which requires the use of Secure Conversation. Secure Conversation enables a consumer and provider to establish a shared security

context when a multiple-message-exchange sequence is first initiated [39]. The web service verifies that the client is a valid account holder. If the condition is met, the authentication web service sends back an authentication confirmation, including a symmetric key established via username and password using HTTPS. Using username authentication security mechanism with a symmetric key signature message level will protect application integrity and confidentiality. For this mechanism, the client does not possess any certificate or key of its own, but instead sends its username/password for authentication. The client shares a secret key with the server. The shared, symmetric key is generated at runtime and encrypted using the service's certificate. The web service client must provide a username and password in addition to specifying the certificate of the web service. To specify the certificate, the client must specify the location and alias to be used of the "truststore" file and its password to identifying the server's certificate [68]. Further information about the IP address and port number to connect to the actual Account Management web service will be sent along with an issued symmetric signature key and session token that identifies the user's unique session. The online stock trading web services authenticate a user with a symmetric signature key, and use a session token to keep track of the user.

2. User account: web service clients can communicate with the User Account Management web service at the IP address and port number returned in Step 1. For each SOAP request, the User Account Management web service will respond with information about the client's account, including current positions and buying power as well as currently open orders. In addition, the service respond

can include portfolio user activities such as Orders, Trades, and a Trade Summary. Once the clients account has been approved and funded, clients will be able to send successful request to the Stock Quote web services.

3. Stock Quoting: web service clients can retrieve stock quotes, users can direct-access trading for options, futures, forex, stocks, bonds and funds. Before users receive real-time data, the user must request it from the *Subscription Account* Management service and retrieve user market data subscriptions. Once the user receives the subscriptions data, the user can add ticker symbols to the stock quote service list. Subscriptions to which they are not subscribed will be sent as delayed data real-time subscription if allowed by the exchange. The ticker is then added to the live stock quote service watch list. A web service client can send requests to the web service and the service responses with live stock quotes.
4. Sending Orders: web service clients can send orders on the tickers that are currently the focus of users' portfolios to the Buy-Sell-Stock web service. By default, the Order entries are parts of user accounts, which also include the watch lists. Web service clients will be able to send request to the Buy-Sell-Stock web service to get live orders, cancel, or complete order requests. In the stock trading process, users may submit, "put" orders, which mean that the clients want to sell stock. Users "buy" orders, which mean that they want to buy stock. The Buy-Sell-Stock web service matches put requests and registers them as successful trades, all request messages must be accompanied by the session key provided to the client in Step 1. The Buy-Sell-Stock web service sends a back order or cancels confirmations, or execution or cancellation reports to the client.

6.4.2 Initiating the Proposed Framework

In order to initiate the testing process of online stock trading web services under test by the proposed framework, the System Analysing Agent (SAA) is activated by receiving an Agent Communication Language (ACL) message with "start SAA" text that initiates the proposed framework.

6.4.3 Initialising Testing the Implemented WS-* Architecture

The SAA retrieves the core functional and QoS system requirements after SAA parsing the WSDL, PBEL, and BPMN 2.0 files. SAA performs data mining supported by machine-learning classifiers from the Learning and Decision Making module and generates the conventional core implementation parameters and variables of Online Stock Trading web services under test, such as, operating methods, agreement binding, message types, service description, service publication and discovery based on core functional standards and specifications in the WS-* Architecture. This data is required to achieve successful test case generation, invocations and interactions between Online Stock Trading web services and services consumers. The Test Execution module which uses TEAs, and the Monitoring module implement the test-harness for Online Stock Trading web services under test, set the input test oracles in the SOAP Body, and execute and monitor the execution of the test case systematically. Simultaneously, machine-learning classifiers from the Learning and Decision Making module support the framewrok modules in real-time. The TEAs carry out further instructions from AA in order to execute tests cases properly.

6.4.3.1 Testing WS-ReliableMessaging, Security, and Addressing

The Authentication web service uses reliable messaging (WS-ReliableMessaging), a security mechanism (WS-Security), and addressing location (WS-Addressing) when collecting QoS policies (WS-Policy) assertions. These QoS policies assertions make mapping environment rules of the Authentication web service protocols during test cases generation and test invocations as shown in Table 6.2.

Services implementation	WSDL Infoset	WS-* architecture specification	Description
Authentication web service	<wsrmp:RMAssertion> <wsp:Policy> <wsrmp:DeliveryAssurance> <wsp:Policy> <wsrmp:InOrder/> </wsp:Policy> </wsrmp:DeliveryAssurance> </wsp:Policy> </wsrmp:RMAssertion> <wsam:Addressing/> </wsp:Policy>	WSDL WS-Policy WS-ReliableMessaging	The web service specifies a policy assertion that specifies that WS-ReliableMessaging protocol must be used when sending messages
	<sp:SignedEncryptedSupportingTokens> <wsp:Policy> <sp:UsernameToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient"> <wsp:Policy> <sp:WssUsernameToken10/> </wsp:Policy> </sp:UsernameToken> </wsp:Policy> </sp:SignedEncryptedSupportingTokens>	WS-Policy WS-Security	The web service specifies a policy that includes WS-SecurityPolicy requires that the token is both signed and encrypted

Table 6.2. Authentication web service environment protocol assertions communications rules mapping

The LDM module makes a decision based on training datasets according to mapping rules in Table 6.2, and preforms data mining by capturing the specific QoS: the WS-Security, WS-Addressing, and WS-Policy protocols data retrieved from the Authentication web services requirement specification in the metadata database, and on that basis the test-harness environment is predicted. Taking WS-Security protocols as an example for making machine-learning test-harness decisions based on training datasets, WS-Security seeks to encapsulate the security interactions described above within a set of SOAP Headers. WS-Security handles credential management and defines a special

element, UsernameToken, to pass the username and password to the Authentication web service. WS-Security also provides a place to provide binary authentication tokens such as Kerberos Tickets and X.509 Certifications BinarySecurityToken [85]. Figure 6.2 shows a section of structure of the decision tree from the WS-Security *test execution* environment training dataset.

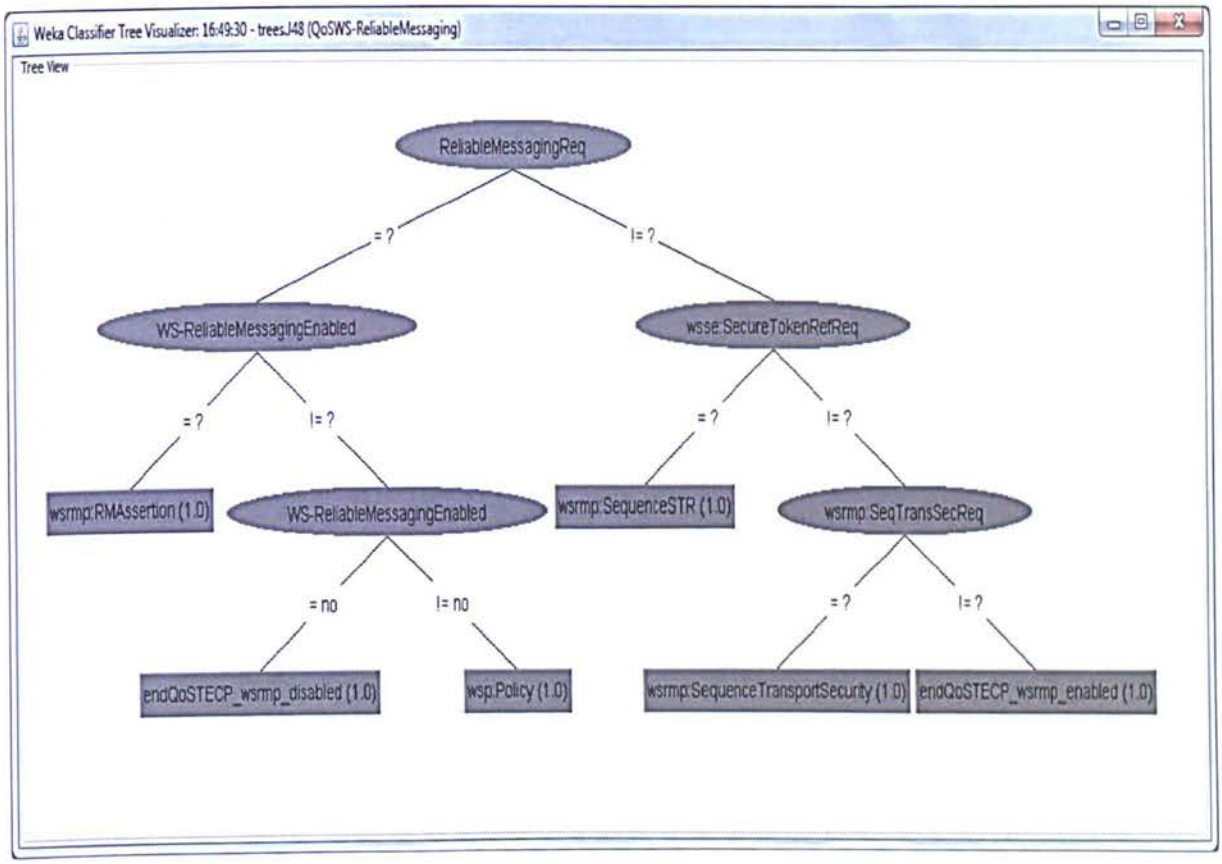

Figure 6.2. A section of the structure of the decision tree from training dataset for the WS-Security test execution environment

Figure 6.3 shows the test-harness environment setup data which is generated as part of the input parameters and header in the SOAP Body in the test cases, which are systematically executed accordingly as the test case execution scenarios.

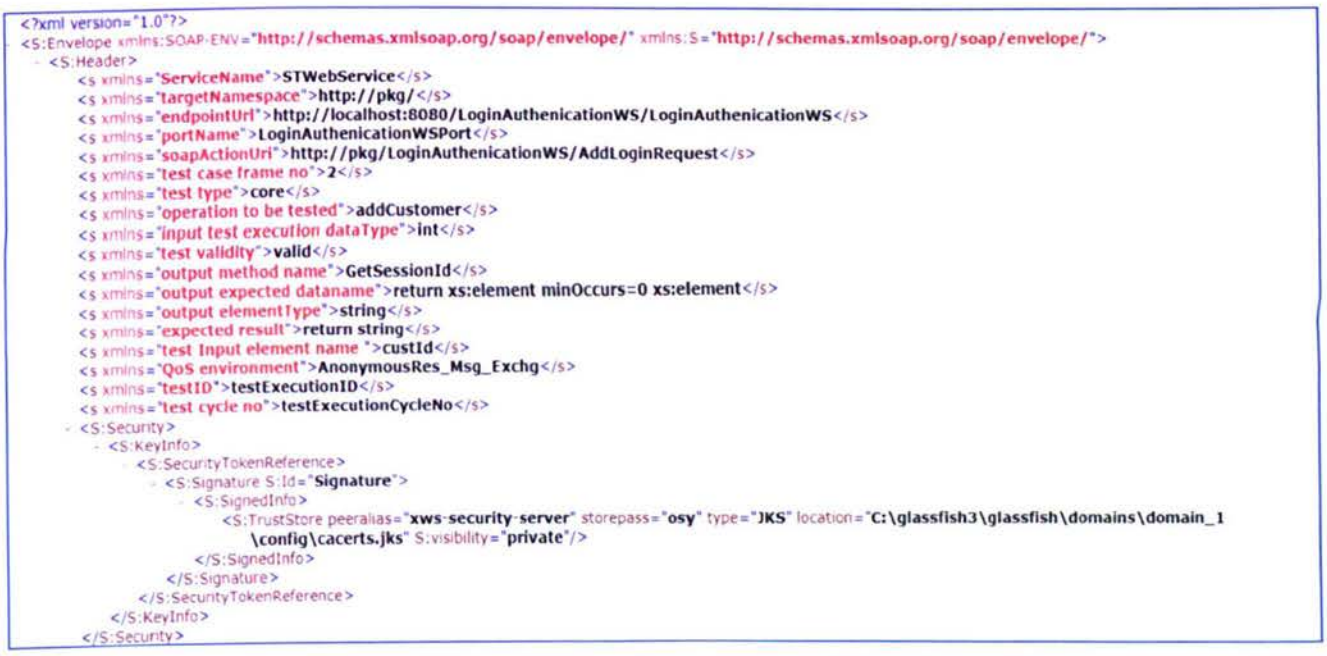

Figure 6.3. A test harness setup scenario for authentication web services

6.4.3.2 Testing Choreography (WS-CDL)

The Subscription Account Management web service uses reliable messaging (WS-ReliableMessaging) with WS-Chorography, and uses addressing location (WS-Addressing) within the collection of QoS policies (WS-Policy) protocols. In WS-Chorography, protocol participants use a multistep process to complete a collaborative SOAP request between multiple-participants. Choreography is a kind of process that focuses on how participants coordinate their interactions through the exchange of messages. At the beginning of the choreography, a web services' client uses the session key returned from the Authentication web service to request from the Subscription Account Management web service to retrieve a ticker subscribing data. WS-Chorography describes interactions between peers without describing how those peers do their business internally. Using BPMN2.0 the process is a graph of flow elements, which are a set of activities, events, gateways and sequence flow. Choreographies represent sets of tasks performed by

participants. A Process describes a sequence or flow of Activities in an organization with the objective of carrying out work. The BPMN2.0 processes of the Subscription Account Management web service are shown in Figure 6.4.

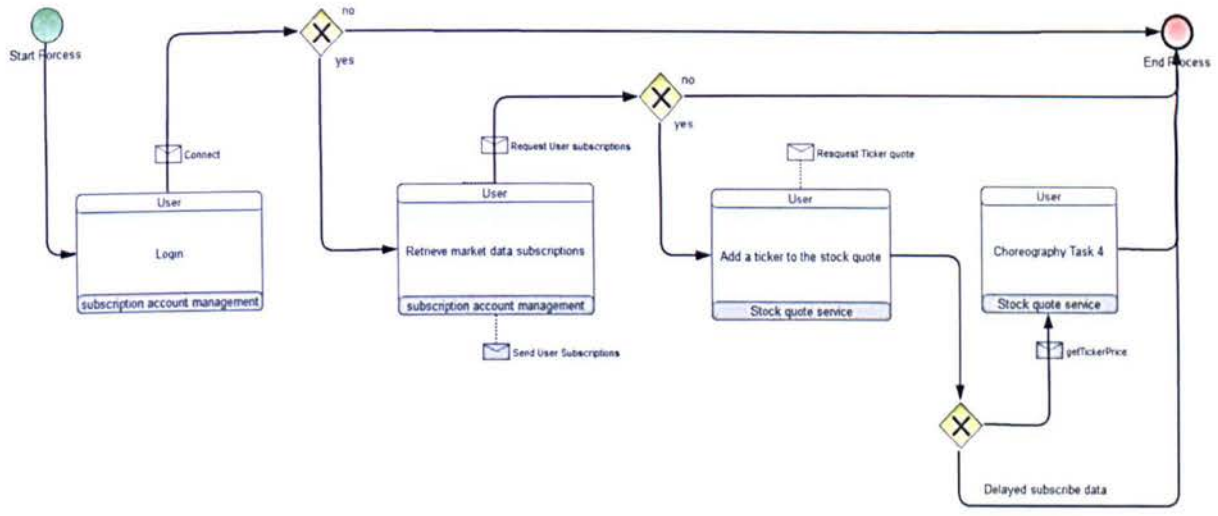

Figure 6.4. The BPMN2.0 processes of the subscription account management web service

The core of a choreography task is depending on the element of choreography, which specifies a set of peer-to-peer interactions. A package can contain one or more choreography assertions, one being the root for all other assertions. These assertions can be mapped as communications rules as shown in Table.6.3.

Services Implementation	WSDL Inforset	WS-*architecture specification	Description
Subscription Account Management web service	<bpmn2:choreographyTask id="*ChoreographyTask_2*" name="*Retrieve market data subscriptions*" initiatingParticipantRef="*_Participant_4*"> <bpmn2:incoming>SequenceFlow_5</bpmn2:incoming> <bpmn2:outgoing>SequenceFlow_13</bpmn2:outgoing> <bpmn2:participantRef>_Participant_4</bpmn2:participantRef> <bpmn2:participantRef>_Participant_5</bpmn2:participantRef> <bpmn2:messageFlowRef>MessageFlow_4</bpmn2:messageFlow Ref> <bpmn2:messageFlowRef>MessageFlow_5</bpmn2:messageFlow Ref> </bpmn2:choreographyTask><bpmn2:message id="*Message_1*" itemRef="*ItemDefinition_1*" name="*Request User subscriptions*"/> <bpmn2:itemDefinition id="*ItemDefinition_1*" structureRef="*ns1_StockTickerPrice*"/> <bpmn2:interface id="*_Interface_3*" implementationRef="*ns1:StockTickerPrice*" name="*StockTickerPrice*"> <bpmn2:operation id="*_Operation_3*" implementationRef="*ns1:getTickerPrice*" name="*getTickerPrice*"> <bpmn2:inMessageRef>_Message_4</bpmn2:inMessageRef> <bpmn2:outMessageRef>_Message_5</bpmn2:outMessageRef>	BPMN2.0 WSDL WS-Chorography WS-ReliableMessaging (In principle) service WS-Addressing WS-Policy	Choreography Task defines messages between the two participants in the task by using connectors, gateways, or events, can create a sequence flow between multiple choreography tasks to represent decision controls and sequence flow of the tasks.

Table 6.3. Subscription account management web service environment protocol assertions communications rules mapping

Using the BPMN2.0 process, rules mapping to transform communication rules from BPMN2.0 to WS-CDL and to WSDL [5] are set (as shown in Table.6.4).

BPMN Element	WS-CDL Infoset	WSDL Infoset
Message	Information Type	Types (schema XSD)
	Variable	Element
	Exchange	Message Input/output action=request and output action=respond.
Participant		Service
	Role Type	operation
	Participant	PortType
MessageFlow	Relationship Type	type
Choreography	Choreography	Interface
ChoreographyTask	Interaction	Operation
ExclusiveGateway	Work Unit	-----
IntermediateThrowEvent	Interaction	Operation
EndEvent	FinalizerBlock	-----
-----	Sequence	-----
EventBasedGateway	Choice	-----

Table 6.4. Transforming and mapping from BPMN to WS-CDL and to WSDL

The LDM module makes a data mining decision based on training datasets as in Table 6.4. Thus, it captures the specific core choreographies assertions of the choreography task,

which specifies a set of peer-to-peer interactions, and on the basis of this the test-harness environment is predicted and requests to the Subscription Account Management web service is established, then a subscribed ticker data can be retrieved automatically and systematically.

6.4.3.3 Testing Web Services Atomic Transaction and WS Coordination

The Buy-Sell-Stock web service uses WS-AtomicTransaction, WS-Coordination, and uses addressing location (WS-Addressing) in collecting QoS policies' (WS-Policy) protocols. The QoS policies assertions can be mapped as communications rules as shown in Table 6.5. To begin an atomic transaction, the web service client firstly locates a WSCoordination coordinator web service that supports WS-Transaction within Buy-Sell-Stock web service. Once located, the client sends a WS-Coordination "CreateCoordinationContext" message to the activation service specifying "http://schemas.xmlsoap.org/ws/2002/08/wstx" as its coordination type and will recveive an appropriate WS-Transaction context from the activation service. The response to the "CreateCoordinationContext" message, the transaction context has its CoordinationType element set to the WS-Atomic Transaction namespace "http://schemas.xmlsoap.org/ws/2004/10/wsat" and also contain a reference to the atomic transaction coordinator endpoint within the Buy-Sell-Stock web service (the WS-Coordination registration service) where participants can be enlisted. After obtaining a transaction context from the coordinator, the client then proceeds to interact with a Buy-Sell-Stock web service to accomplish its business-level work. With each invocation of a Buy-Sell-Stock web service, the client add requests the transaction context into a SOAP header block, such that the each invocation is implicitly scoped by the transaction.

Services Implementation	WSDL Inforset	WS-*architecture specification	Communications rules descriptions
Buy-Sell-Stock web service	wsp:ExactlyOne> <wsp:All> <wsat:ATAssertion wsp:Optional="true" </wsp:All> </wsp:ExactlyOne> </wsp:Policy>	WSDL WS-AtomicTransaction WS-Policy	The web service specifies a policy expression containing the Atomic Transaction policy assertion.
	< portType> <wsdl:portType name="ActivationCoordinatorPortType"> <wsdl:operation name="CreateCoordinationContext"> <wsdl:input message="wscoor:CreateCoordinationContext"/> </wsdl:operation> </wsdl:portType> <portType> <wsdl:portType name="ActivationRequesterPortType"> <wsdl:operation name="CreateCoordinationContextResponse"> <wsdl:input message="wscoor:CreateCoordinationContextResponse"/> </wsdl:operation> <wsdl:operation name="Error"> <wsdl:input message="wscoor:Error"/> </wsdl:operation> </wsdl:portType>	WS-Coordination Activation Service	The web service defines a single port declaring CreateCoordinationContext operation. The operation takes an input to specifying the details of the transaction to be created; It returns an output containing the details of the newly created transaction context: the transaction identifier, coordination type, and registration service URL.
	<portType> <wsdl:portType name="RegistrationCoordinatorPortType"> <wsdl:operation name="Register"> <wsdl:input message="wscoor:Register"/> </wsdl:operation> </wsdl:portType> <!-- Registration Requester portType Declaration --> <wsdl:portType name="RegistrationRequesterPortType"> <wsdl:operation name="RegisterResponse"> <wsdl:input message="wscoor:RegisterResponse"/> </wsdl:operation> <wsdl:operation name="Error"> <wsdl:input message="wscoor:Error"/> </wsdl:operation> </wsdl:portType>x	WS-Coordination Registration Service	The web service defines registration service message targets the CoordinationContext, and provides the name of the protocol it wants to register for and the Participant service's EndpointReference. The RegisterResponse message provides the AtomicTransaction service's EndpointReference.

Table 6.5. Buy-Sell-Stock web service environment protocol assertions communications rules mapping

6.4.3.4 Testing Orchestration (WS-BPEL)

The User Account Management web service uses the BPEL web services to control the activities of the User Account Management web service via scopes, structured and basic activities. Unlike BPMN2.0, using BPEL, the SOAP request begins with business logic of application learning the method and parameters to call web services methods. Also, each activity allows for nesting of other activities.

6.5 Test Effort Measurement

Test effort or the test cost measurement is carried out to estimate cost in time and machine resources for test-case generation, test-execution, and collecting and logging the test result. The test cost is estimated by measuring the actual time required to generate and execute test cases along the test steps presented as break down parts of the functional modules used by the proposed framework, in particular, for evaluating the test cost for testing ClientsLogin method of the Authentication web service. Test cost in time is collected for the generation test case with test input of values as outlined in Table 6.6, including test harness setup which is described in Table 6.2, while applying the assertions communications rules for mapping establishing authentication web service invocation, this includes symmetric key information with *username and password using* HTTPS.

6.5.1 Test Case Generation Result

The Test Case Generation module captures and identifies the data from the results of data mining of the core and the QoS requirements specification of the online stock trading web services system by implementing the CP and ECP procedure as proposed in Section 5.3.1 and 5.3.2. The result of test cases generation of the online stock trading web services methods is shown in Table 6.6.

Service	WS-* Architecture specification	Web service method	# NoTC
Authentication web service	WS-ReliableMessaging, WS-Security, WS-Addressing , and WS-Policy protocols	ClientsLogin method with 2 inputs parameter with 6 input values (2^6) *1	64
Account Management web service	WS-BPEL	GetUserPortfolio GetBuyingPower GetOpenOrders AddFund	256
Subscription Account Management web service	WS-ReliableMessaging, WS-Chorography, WS-Addressing , and WS-Policy	GetsTickerSubscription	64
Buy-Sell-Stock web service	WS-AtomicTransaction, WS Coordination, WS-Addressing, and WS-Policy	GetLiveOrder GetCancelOrder GetCompleteOrder PutOrder BuyOrder GetReport	384

Table 6.6. Test case generation result for online stock trading *web services* methods

6.5.2 Test Effort Result

Figures 6.5 and Figures 6.6 show the results of the testing task expended by the proposed framework on generating test cases of the Authentication web service. In the graphs, the vertical axis shows the breakdown of the functional modules used by the proposed framework, while the horizontal axis shows the work-time consumed (total actual time in seconds) in the test-case generation, in the test-harness implementation, which includes test environment setup, and test execution cases generation. The results demonstrated testing effort (test cost) using the proposed framework. The actual results show that the proposed framework required approximately 20.67 seconds for System Analysis and Test Cases Generating steps for all test cases for the Authentication web service per test cycle.

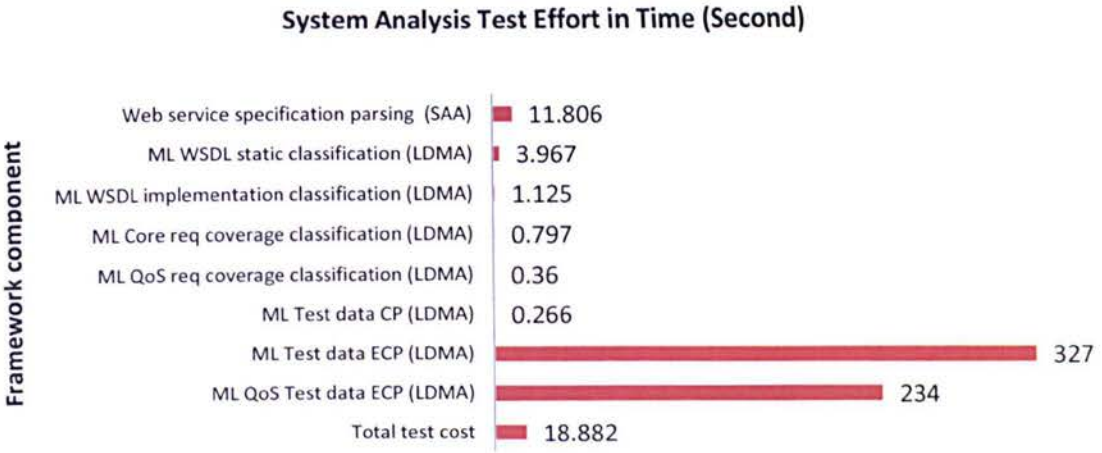

Figure 6.5. Test cost results of testing task for system analysis step of the authentication web service

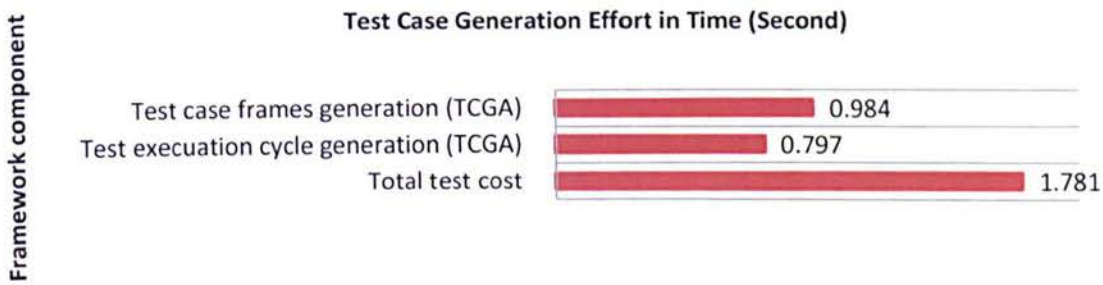

Figure 6.6. Test cost *results of* testing task for test case generation step of the authentication web service

6.5.3 Experimental Setup

The use case prototype is built with Java 7.0 (Java SDK 1.7) and Maven 3.0., it is designed to be run on a Glassfish server Platform. The computer used to run the use case is configured as follows: Intel Core i3 2.4 GHz with 6GB RAM, Windows 7 Professional, NetBeans 7.4.0, and MySQL 5.2. The proposed framework ran against the use case prototype web services on the same environment.

6.5.4 Computer Resource Test Cost

A resource usage measurement is carried out to see how much computer resource tests cost in the running agents of the proposed framework. The Computer Performance Monitoring application is used to measure CPU and memory usages aspects according to selected counters of the computer resources specifically to measure: Processor Usage In Time (CPU), Committed Bytes of In Use (Memory), and Available Bytes In Use (Memory) of the proposed framework computer resource usage for testing Authentication web service use case. The result of the machine test cost is shown in Figure 6.7. The figure displays CPU and memory usages of testing task for System Analysis step which shows the process which used by the testing agents consuming maximum 23.3% of processor time for a duration of 1.4 seconds and with an average of 1.5 % during the process of testing the Authentication web service.

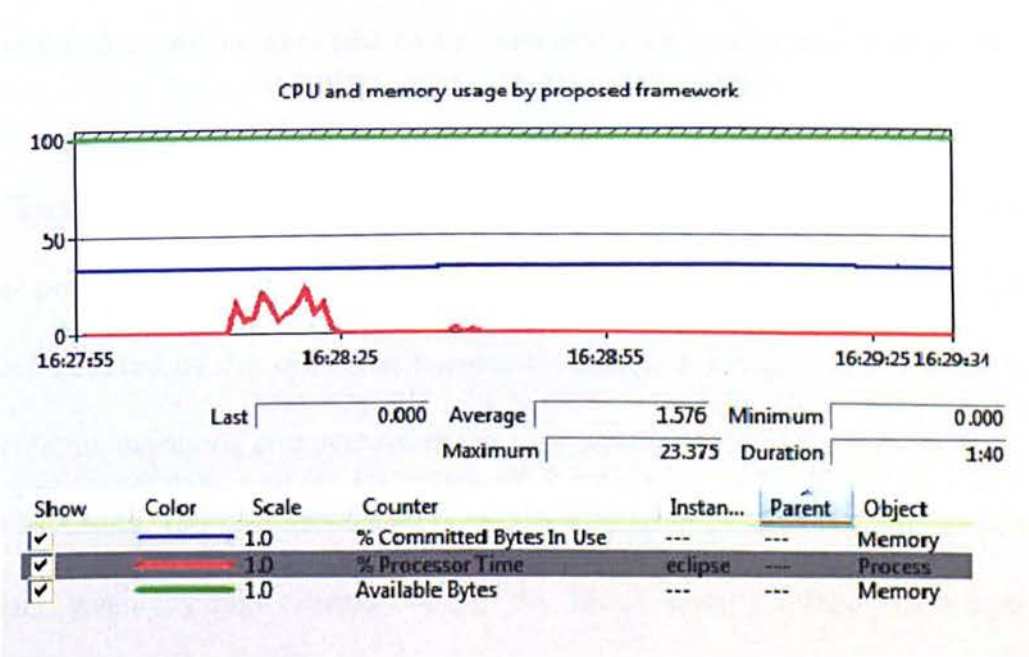

Figure 6.7. CPU and memory test cost results of testing task for system analysis step of the authentication web service

The result of the machine test cost for the Test Case Generation step is shown in Figure 6.8 which displays CPU and memory usages of the testing task. The figure shows the process which is used by the testing agents consuming with maximum 50% for a duration of 1.4 second during the process of the testing task of Authentication web service.

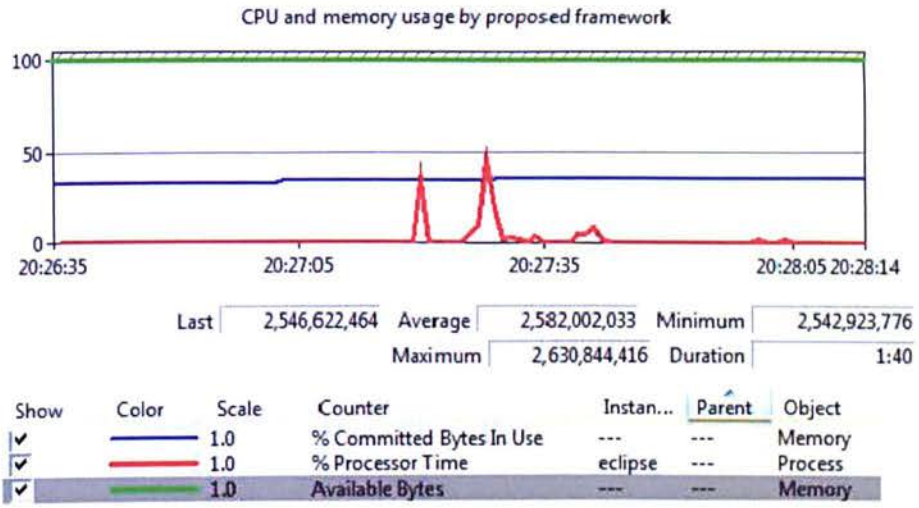

Figure 6.8. CPU and memory test cost results of testing task for test case generation step of the authentication web service

6.5.5 Test Effectiveness Measurement

Another practical evaluation is carried out to measure the test effectiveness of the test approach adapted by the proposed framework. The test effectiveness measurement is conducted by measuring and evaluating the cost-effectiveness of the authentication web service test suite. The cost-effectiveness was evaluated by computing the test suite (test cost) for a given coverage criterion against the defect detection adequacy criteria. The tests cases were based on core functional and QoS system requirements, and the test oracles were based on data-type references. In the following subsections, the test effectiveness of proposed framework will be empirically evaluated.

6.5.5.1 Test Coverage Evaluation

A test coverage analysis was carried out to evaluate the coverage level achieved by the Authentication web service method source code. This analysis was conducted in a very similar fashion to that of the empirical test coverage evaluation in Chapter 5. The test coverage analysis is implemented by generating unit test suite and instrumenting the code source, the JUnit unit testing tool is used to generate the Unit Test Cases (UTC) which includes the same combination of input test data, methods calls, and expected outputs which are used during test-case generation through the proposed framework. As mentioned earlier in Chapter 5, the CodeCover tool is used to instrument and gauge the coverage level achieved by a test suite for each unit test case, which provides the percentage of the proportion of requirements that have been satisfied for statement (instruction) coverage, branch coverage, line coverage, complexity coverage, and method coverage .Using Cartesian product (as exemplified in Section 5.3.2), every test input of a unit test case is paired with every test input of every other unit test case. The Authentication web service is composed of test cases for testing with two input parameters of a String. Thus, all combinations of the test inputs across all unit tests are obtained, 36 (6^2)*1 of test cases are identified and generated. The JUnit tool was employed to execute the test suite, accordingly, by using the CodeCover tool to generate the coverage rate percentages covered by each unit test. The graph in Figure 6.9 summarises the result of test suite coverage.

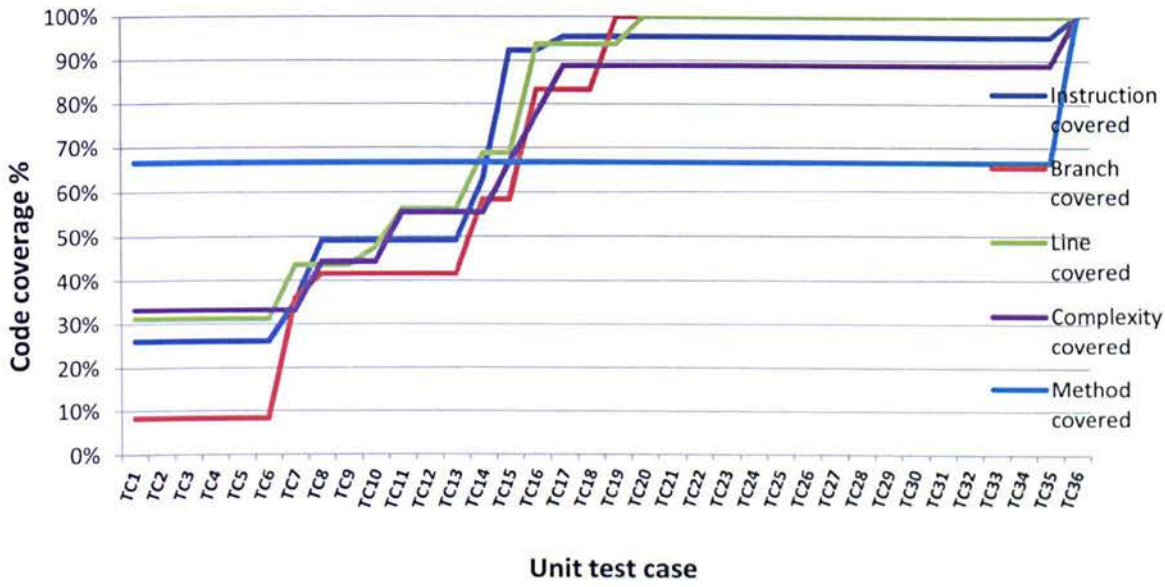

Figure 6.9. The result of the test suite coverage

The graph in Figure 6.9 shows that as TEA executed the 36 unit test cases one by one, the code coverage is improved, reaching 100% coverage, for all, i.e. for instruction, branch, line, complexity and method coverage percentage value. This result is consistent with earlier result obtained through the empirical test coverage evaluation that conducted in Chapter 5. Hence, the result of this test coverage evaluation demonstrates full test coverage criteria according to the Authentication web service system requirements.

6.5.5.2 Defect Detection Effectiveness Measurement Evaluation

A mutation-detection effectiveness measurement is carried out to evaluate the defect detection level which is achieved by the Authentication web service test suite. Figure 6.10 summarizes in a graph the mutation-detection effectiveness of the generated test suite by the proposed framework. The horizontal axis signifies applied mutant types, while the vertical axis signifies the percentage of the test cases which detected the mutants. The graph illustrates that the test approach adapted by the proposed framework is capable of

finding all defects of the selective mutation types which are applied on the Authentication web service system requirement with 100% mutation score.

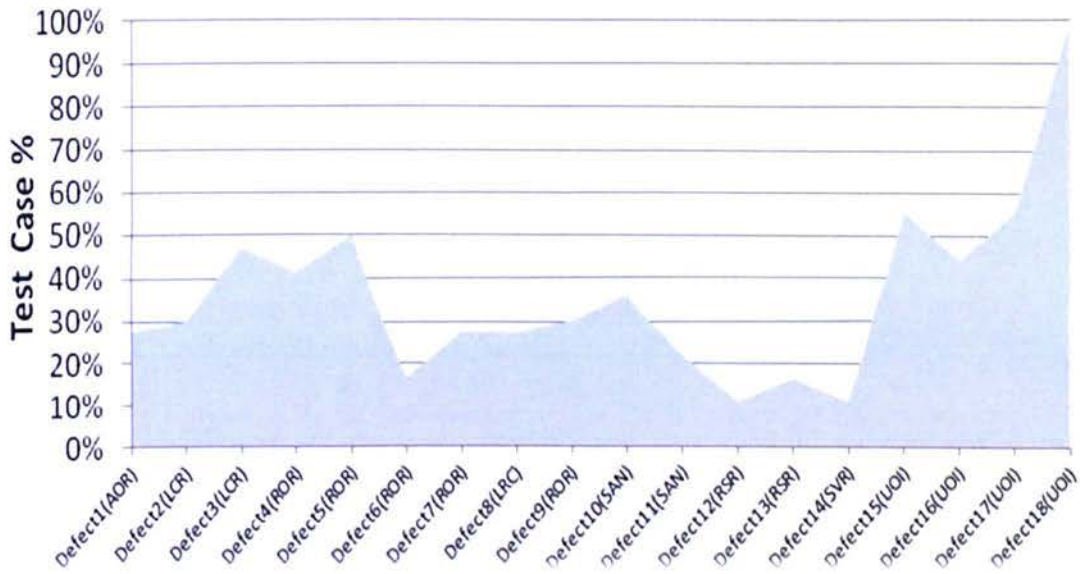

Figure 6.10. Defect detection ratio for authentication web service test suite

6.5.6 Cost-Effectiveness Measurement Evaluation

The cost effectiveness measurement was calculated and plotted against the test suite size (test cost). The approach is adapted by calculating and plotting the defect detection effectiveness against the test cost. The resulting graph is presented in Figure 6.11, where the horizontal axis represents the test suite size or test cost, while the vertical axis represents the detected defect rates. The figure shows the cost-effectiveness line which illustrates as the test suite size increases, the number of detected defects which provide evidence of the defect detection adequacy criteria, this indicates that the actual test suite cost effectiveness at detecting defects. The result of the cost effectiveness measurement demonstrates that the proposed framework is consistent which indicates high cost-effectiveness from the perspective of the test coverage.

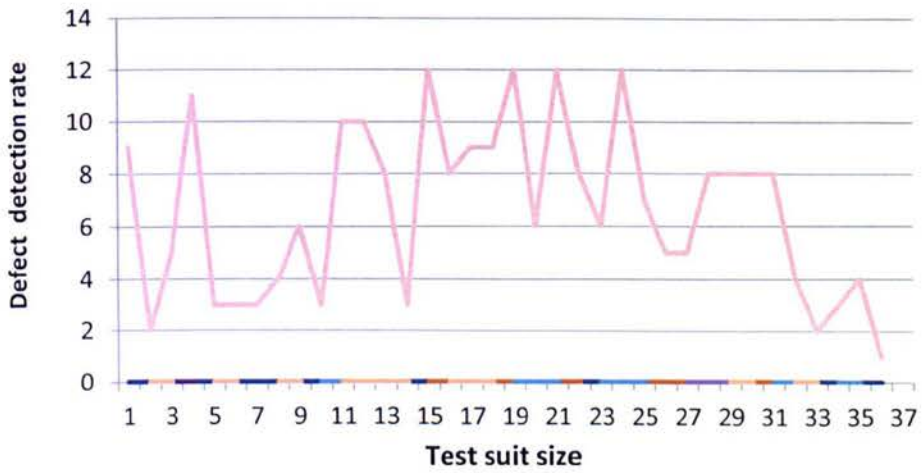

Figure 6.11. Cost-effectiveness measurement of the authentication *web service* test suite

6.6 Conclusion

An online stock trading use case is developed and used as a prototype system by means of a proof-of-concept implementation for testing systems based on SOA testing using the proposed framework solution. The proposed framework is utilised on a larger scale as an approach for improving the cost-effectiveness of various testing stages. The framework is extended to support test oracle generation from web services business process XML languages, Business Process Execution Language (BPEL), and Web Services Choreography Description Language (WS-Choreography) using BPMN2.0 processes. The test cases generated from this specification will drastically reduce the time it takes to generate test cases and test the SOA based applications and will improve the Return of Investment (ROI). This specification addresses all kinds of collaborations, especially dynamically invoked and freely interacting different types of web services components. The specifications implemented are in state-full e-commerce system, in the B2B world, where services are invoked dynamically according to the result of the previous

invocation. The evaluation demonstrated the benefits of the proposed framework as an automated framework with low cost associated which gives confidence to the research outcomes. Furthermore, the evaluation analysis showed that the proposed framework can support automated QoS test generation and execution with high-test coverage for advanced and complex SOA implementation within industry sectors.

Chapter 7 – Conclusion and Future Work

7.1 Conclusion

SOA has become popular and gained significant attention and support from major companies in computing. However, testing the implementation of systems based on SOA such as web services has become a major issue, as the task of testing is made more complex by the specific characteristics of these systems. The primary aims and contributions of this thesis are the design, implementation, and evaluation of a new automatic machine-learning framework by means of a proof-of concept implementation for testing system based on SOA. The work which was carried out to achieve these aims, as well as the results of the work, was reported in the previous chapters of this thesis. In this section the achievements and conclusions, which have been previously drawn, will be summarised.

The research began with a thorough investigation into the testability problem of SOA based systems which has become a major issue, in particular the systematic validation and testing of SOA components. The investigation is considered a necessary prerequisite to support intelligent learning of SOA component testability which facilitates test environment knowledge acquisition and helps SOA component test engineers and users to obtain high testability design knowledge of functional and non-functional requirements so that component test criteria is easily scoped and effective testing accomplished.

The thesis then conducted a thorough investigation and evaluation of the existing contributions presented in the literature. The investigation started by discussing and comparing the SOA testing frameworks and tools, sharing common aspects, and

evaluating their approaches into a means of improving the testability of systems based on SOA, as well as their deficiencies and shortcomings with respect to current testability issues which are due to specific SOA characteristics. The actual outcome of the investigation and evaluation from the literature has appealed to reconsider and redesign the current traditional and automated testing approaches, and to invent new testing approaches and frameworks. For that matter, the thesis has investigated a suitable approach with test coverage strategies in order to specify how it is implemented with different type of approaches, methods, and computational techniques and to provide a practical solution of testing these systems. Hence, the thesis highlighted a suitable and ideal approach to enabling an effective testability degree of SOA systems based on combining automated test simulators as systematic offline testing with online testing and monitoring for validating and verifying the core system and service's trustworthiness, based on protocols and standard requirements and test creation and execution coverage analysis. Moreover, the thesis highlighted a suitable machine-learning approach which can automatically derive skeletons of SOA test cases and provide support for their execution and result analysis.

The identified testability issues of SOA based system have been addressed in this thesis by designing, developing, and implementing an automated and systematic testing and monitoring framework as a functional prototype and as actual deliverable for the industry. The framework is supported by MAS and Machine Learning techniques and based on knowledge discovery and data mining of protocols and standard requirements based on web service protocol stack and test coverage analysis. The functional prototype provides a practical solution to the testability problems in SOA systems by automatically

establishing testability links between the prerequisites of intelligent knowledge of SOA testability and system under test requirement and test coverage analysis.

The design of the functional prototype aimed to combine existing computational techniques and methods for resolving the problems of SOA testability and to improve testing systems using top-down testing approach which creates necessary inputs and output required from high-level system specifications based on WS-* Architecture for each method in each service within SOA system ,and then invoke the method which enables an effective testability degree based on combining automation machine-learning with test cases generation and monitoring their execution and resulting analysis.

Moreover, the aspects of the design and implementation of the functional prototype of the machine-learning SOA testing framework was elaborated and demonstrated together with all the required functional modules and functional flow processes within the framework.

Furthermore, the novel framework on which it has evolved has demonstrated the advantages of utilising the MAS approach supported by intelligent reliability, such as preferences, with purely SOA principles and a standard-based approach using the web services protocol stack. The identified approach is considered essential to increase the QoS and level of deployment within both academic and industry sectors. In addition, the proposed framework has demonstrated a suitable level of achievement of using different type of methods and computational techniques to develop an automated testing case generation module supported by the LDM module to apply a structural machine-learned technique and to process knowledge discovery and data mining of the core and QoS requirements based specification and then automats black-box test cases generation

supported by MAS technique. In addition, the proposed framework has demonstrated that the machine-learned data mining method produces correct classification decisions based on different type training datasets which applied upon new classification cases. Furthermore, the framework has demonstrated a suitable level of achievement of developing an automated and intelligent test case-coverage analysis supported by machine-learning methods applied to determine if the generated test suite satisfies the coverage criteria according to the core and QoS requirements specification of the web services under test. Moreover, the novel framework has demonstrated achieving the *processing* test execution simulation tasks using MAS agents as test simulators and by performing offline testing with online testing and monitoring for *validating and verifying* server response analysis, and feeding the test executions results into a test database. The identified test execution data include the environment settings, which are required for implementing the test harness of the test client at the time of creation of the SOAP message communication dispatcher. The test simulation outputs were used by the LDM module in the learning, reasoning, and decision making process, throughout the test validation procedure within the monitoring process. The novel framework also has demonstrated intelligent analysis achievement of using the feedback from the LDM module to support the learning and decision making concept and produce effective quality-level testing in accordance with the expected test outputs. The proposed framework demonstrated that the LDM module supported by machine-learning classifiers can be trained and then classify the output data of test executions of the baseline test cases, then classify the condition of the test cases, and determine whether the test cases have meet the test objectives.

After detailed presentations of individual modules of the novel framework on which it has evolved and in order to exemplify the practical solution and prove the testability aspects were considered, the thesis has presented the individual modules of the framework together with working examples using practical SOA web services prototypes example implementations. The example implementation results have provided analytical evaluation descriptions of the effective practical evaluation study of the functions of each of the core modules in the proposed framework.

Furthermore, an extensive empirical evaluation of the framework's functional prototype using practical examples based on the quantitative data analysis of cost-effectiveness was carried out, in order to evaluate and prove that significant savings in time and effort and can be achieved by employing the developed framework. The empirical evaluation of the proposed framework is conducted by comparing it to leading commercial and open source benchmark frameworks and tools for testing SOA systems by using key factors such as test cost, defect detection effectiveness and cost-effectiveness measurements.

The test cost carried out as a measure for the test cost and to evaluate the test suite completeness generated by the proposed framework. The test suite completeness result has shown full code coverage by the test suite. This implies that the test suite generated by the proposed framework has a high degree of test cost-effectiveness traced to the requirement specification of the SOA system under test. Moreover, an internal validity factor for test cost evaluation was established and determined through a practical comparison of the proposed framework against leading and comprehensive open source and commercial SOA test tools. The results demonstrated reduced test costs using the proposed framework compared with other SOA test frameworks and tools. The actual

results showed that the proposed framework positively required the lowest amount of time of these SOA test tools. In addition, another internal validity factor which is the degree level of automation, was also established and determined through empirical comparison of the proposed framework against the other SOA test tools according to the automation validity factors. The results showed that the proposed framework could achieve a good degree of automation in comparison to other SOA test tools.

Moreover, an external validity factor which could limit the ability to generalise the proposed framework for supporting SOA industry practices was also determined through an empirical comparison of the level of implementation of these factors against similar frameworks and tools from the literature. The results showed that the highest level of implementation of these characteristics was achieved by the proposed framework in comparison to other frameworks from the literature. This verifies respectively that the proposed framework implements and supports the standards and protocols that make it more transparent and applicable in the implementation according to industry practices. Additionally, another practical evaluation carried out to measure another key factor, the defect detection effectiveness which has showed that the test approach adapted by the proposed framework is efficient and capable of finding all injected mutant types with 100% mutation score, which indicates a high degree of defect detection. A final measure of the cost-effectiveness of the proposed framework was computed. The empirical analyses of the cost-effectiveness of the proposed framework has shown that the proposed framework is effective at a much lower test cost than other SOA test tools. The evaluation demonstrated the benefits of the proposed framework as an automated framework, compared to the cost associated with others which gives credence to the

research outcomes. Furthermore, the evaluation analysis showed that the proposed framework can support automated QoS test generation and execution with high test coverage and defect detection levels, as compared with other frameworks from the literature.

Finally, a use case study conducted experimentally to evaluation effectiveness of the proposed framework by practical and systematic implementation on a generic and complex business use case within industry sectors. The implementation involved an Online Stock Trading web services system which was designed and developed as a prototype for the SOA web services environment implementation for testing systems based on using the proposed framework solution. The use case environment has deployed the majority of the predominant web services protocol stack WS-* Architecture specifications such as WS-Transactions, WS-Security, WS-Chorography, WS-BPEL and other standards and specifications. The deployed WS-* Architecture specifications addressed all kind of collaborations, especially dynamically invoked and freely interacting different types of web services components. The specifications implemented are in state-full e-commerce system, in the B2B world, where services are invoked dynamically according to the result of the previous invocation. The evaluation demonstrated the benefits of proposed framework as an automated framework with low cost associated which gives confidence to the research outcomes. Furthermore, the evaluation analysis showed that the proposed framework can support automated QoS test generation and execution with high-test coverage for advance SOA implementation within industry sectors.

7.2 Future Work

In this section, suggestions are given about how the work presented in this thesis can be further elaborated. The future work can be classified into a three key areas of improvement.

The first improvement for future work is related to the System Analysis process for supporting system requirements based on the web service protocol stack and test coverage analysis of the proposed framework, which to the fact that the current implementation of the system analysis stage does not support function knowledge and constraints based on SOA service method's functional requirement. Such constraints should ideally be expressed as business rules. Business rule knowledge can be derived from business logic e.g. (Integrity Rules—e.g. received request data validation rule, or Derivation Rules—e.g. price calculation rule, Reaction Rules—e.g. action rules for checking if flights where found otherwise skip) which can be used to evaluate the business rules of a SOA services. One of the most important facts about business rules is that they are declarative statements, they specify what has to be done and not how it is to be done. The framework will be extended to support test function knowledge and constraint generation from web services business processes XML languages, such as Business Process Execution Language (PBEL) and Web Services Choreography Description Language (WS-Choreography). This can be achieved by using structural machine-learning technique, to process knowledge discovery of business logics which are embedded inside PBEL or BPMN documents of SOA systems, this can be investigated and integrated into the data mining process of the SOA services under test.

The second improvement for future work is related to LDM module in the learning, reasoning, and decision-making process, throughout the test cases generation. The fact that the current implementation of the machine learning classification method of generation test oracles cannot learn what input or output properties are potentially of interest as true oracles, but only which ones matter once they are defined as primitive type values provided by programming languages which are produced by the ECP method for partitions of choices of the input-output of test data. In other words, without some additional guidance, the learning algorithm is unlikely to find the precise conditions under which test oracles in the test cases. This guidance comes in the form of choices or constraints, as acquired by CP and ECP methods from business logics. As previously described in the first improvement, once the initial knowledge from data mining of the business logics are transformed into abstract test oracles in the generated test cases.

The third improvement is related to the second stage of the SOA testing process in the proposed framework, as the current implementation of the test case generation stage does not support setting test cases update rules. These rules are in turn analysed using LDM module in the learning, reasoning, and decision-making to determine potential improvements of the test suite e.g. redundant test cases, need for additional test cases as well as improvements of the CP and ECP methods , e.g. need to add a category or choices. The result from the analysis will evaluate the effectiveness of the test suites and CP and ECP methods specifications created and trained by a LDM module classifier. The machine learning iterative process can improve the CP and ECP methods specifications to a level that is equivalent to what an expert system would likely produce improvement cycles. The resulting test suites will be more effective in terms of defect detection.

Finally, the fourth suggested improvement for future work is related to the general SOA testing process of the proposed framework. The framework will be extended to support Reverse Engineering Modeling of the specific SOA system under test. In practice, the system specification is used to identify the test cases. In this case, the test specification has to be either reverse-engineered or created from high-level system specifications, for example the WSDL document. To enable reverse engineering to learn and conclude the SOA system under test specifications, the output domain of the generated test suite of the SOA system has to thoroughly exercise a draft model of the SOA system under test according to various core functional, QoS test environments and coverage metrics form the test suite to produce a contract of SOA services specification, e.g. WSDL document. The resulting WSDL document is fed then into framework testing process which automatically generates a test suite. The generated test suite thoroughly exercises the test model by comparing the outputs produced by the SOA system under test and the model on the tests in the test suite. The deviations in the behaviour of the model from the SOA services specification under test are readily detectable and can be used to guide the user in refining the model to ensure that it correctly captures the behaviour of the SOA system.

References

[1] Abbas Tarhini, Hacène Fouchal, and Nashat Mansour. 2005. A simple approach for testing web service based applications. In Proceedings of the 5th international conference on Innovative Internet Community Systems (IICS'05), Alain Bui, Marc Bui, Thomas Böhme, and Herwig Unger (Eds.). Springer-Verlag, Berlin, Heidelberg, 134-146. DOI=10.1007/11749776_12 http://dx.doi.org/10.1007/11749776_12.

[2] Aberdeen group: SOA and Web Services Testing: How Different Can It Be?, August 2007. Retrieved from: http://www.aberdeen.com/Aberdeen-Library/4117/RA-soa-web-services.aspx.

[3] Active vs. Passive Web Performance Monitoring. Retrieved from: https://www.dotcom-monitor.com/release-active-vs-passive-web-performance-monitoring.aspx.

[4] Agitar:http://www.agitar.com.

[5] Alahmari,S., A design framework for identifying optimum services using choreography and model transformation.2012).University of Southampton, Faculty of Applied Science, Doctoral Thesis.

[6] AppLabs.com :Approach to Testing SOA Applications. App_WhitePaper_Approach_to_SOA_1v04, Available from :http://www.docstoc.com/docs/4248918/Approach-to-Testing-SOA-Applications,2007.

[7] Architecture Maturity Models, 2006 , TOGAF- the Open Group, http://pubs.opengroup.org/architecture/togaf8 doc/arch/chap27.html.

[8] Ariba, International Business Machines Corporation, Microsoft. Retrieved from: http://www.w3.org/TR/wsdl.

[9] Bartolini, A.Bertolino, S.Elbaum and E.Marchetti, "Whitening SOA Testing," Proc. 7th joint meeting of the European Software Engineering.

[10] Bertolino, A. 2009. Approaches to testing service-oriented software systems. In Proceedings of the 1st international Workshop on Quality of Service-Oriented Software Systems (Amsterdam, The Netherlands, August 24–28, 2009). QUASOSS '09. ACM, New York, NY Bertolino, A. 2009. Approaches to testing service-oriented software systems. In Proceedings of the 1st international Workshop on Quality of Service-Oriented Software Systems (Amsterdam, The Netherlands, August 24–28, 2009). QUASOSS '09. ACM, New York, NY.

[11] Bertolino, A., L. Frantzen, A. Polini and J. Tretmans, Audition of web services for testing conformance to open specified protocols, in: R. Reussner, J. Stafford and C. Szyperski, editors, Architecting Systems with Trustworthy Components, number 3938 in LNCS (2006).M. Young, The Technical Writer's Handbook: Univ. Science, 1989.

[12] Bertolino, A.Angelis, G. DeFrantzen, L.Polini, A.Suzuki., K, Model-Based Generation of Testbeds for Web Services, Testing of Software and Communicating Systems : 20th IFIP TC 6/WG 6.1 International Conference, TestCom 2008 8th International Workshop, FATES 2008 Tokyo, Japan, June 10-13, 2008 Proceedings.

[13] Bloomberg,J., "Principles of SOA". Retrieved from:http://adtmag.com/articles/2003/02/28/principles-of-soa.aspx, 1105media.com 02/28/2003.

[14] Borland SilkPerformer SOA edition:http://www.borland.com/us/products/silk/silkperformer.

[15] Bozkurt, M., Harman, M. and Hassoun, Y. (2013), Testing and verification in service- oriented architecture: a survey. Softw. Test. Verif. Reliab., 23: 261–313. doi: 10.1002/stvr.1470.

[16] Briand, L.C.; , "A Critical Analysis of Empirical Research in Software Testing, " Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on , vol., no., pp.1-8, 20-21 Sept. 2007.

[17] Bucchiarone, A.,Melgratti, H., Severoni, F.: Testing service composition. In Proceedings of the 8th Argentine Symposium on Software Engineering (ASSE'07). (2007).
[18] Canfora, G.; Di Penta, M.; , "Testing services and service-centric systems: challenges and opportunities, " IT Professional , vol.8, no.2, pp.10-17, March-April 2006 doi: 10.1109/MITP.2006.51.
[19] Cesare Bartolini, Antonia Bertolino, Eda Marchetti, Andrea Polini, "WS-TAXI: A WSDL-based Testing Tool for Web Services," Software Testing, Verification, and Validation, 2008 International Conference on, pp. 326-335, 2009 International Conference on Software Testing Verification and Validation, 2009.
[20] Chan, W. K., Cheung, S. C., and Leung, K. R. 2005. Towards a Metamorphic Testing Methodology for Service-Oriented Software Applications. In Proceedings of the Fifth international Conference on Quality Software (September 19 - 20, 2005). QSIC. IEEE Computer Society, Washington, DC, 470-476. DOI= http://dx.doi.org/10.1109/QSIC.2005.67.
[21] Chatterjee, J., Testability, StickyMinds.com. Retrieved from:http://www.stickyminds.com/sitewide.asp?ObjectId=8077&Function=edetail&ObjectType=ART, 2010.
[22] Chu, M., Murphy, C., and Kaiser, G. 2008. Distributed In Vivo Testing of Software Applications. In Proceedings of the 2008 international Conference on Software Testing, Verification, and Validation (April 09 - 11, 2008). ICST. IEEE Computer Society, Washington, DC, 509-512. DOI= http://dx.doi.org/10.1109/ICST.2008.13.
[23] CodeCover: http://codecover.org/.
[24] Constant Field Values, Java 2 Platform Standard Edition 5.0 API Specification. Retrieved from:http://download.oracle.com/javase/1.5.0/docs/api/constant values.html.
[25] Demillo, R.A., Guind.S, king, K.N., Mccrackn, W,.M., And Offutt, A, .J.1988. An extended overview of the Mothra software testing environment. In Proceedings of the 2nd Workshop on Software Testing, Verification, and Analysis (Banff, Alberta, Canada, July).IEEE Computer Society Press, Los Alamitos, Calif., 142-151.
[26] Developing Cost-effective Model-based Techniques for GUI Testing, Xie, Q., University of Maryland, College Park, ISBN 9780542961168, http://books.google.ie/books?id=t2wkmdNGWQ4C, 2006, University of Maryland, College Park.
[27] DoD Towards Software", Wipro IT Business. Retrieved from: http://www.wipro.com/wiproforms/thankyou.aspx?ReturnUrl=/datadocs/whitepaper/wipro_soa_testing.pdf,2009.
[28] Dung Cao, Richard Castanet, Patrick Felix, Kevin Chiew. An Approach to Automated Runtime
[29] Eclipse documentation Archived Release, from: http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.jst.ws.doc.user%2Fconcepts%2Fcws.html
[30] El Yamany, H.F.; Capretz, M.A.M.; Capretz, L.F.; , "A Multi-Agent Framework for Testing Distributed Systems," Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International , vol.2, no., pp.151-156, 17-21 Sept. 2006
doi: 10.1109/COMPSAC.2006.98
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4020160&isnumber=4020118.
[31] Eler, M.M.; Delamaro, M.E.; Masiero, P.C., "Using structural testing information to support monitoring activities," Service Oriented System Engineering (SOSE), 2011 IEEE 6th International Symposium on , vol., no., pp.25,30, 12-14 Dec. 2011 doi: 10.1109/SOSE.2011.6139089.
[32] Erl,T.,"What Is SOA". Retrieved from:http://www.whatissoa.com/p9.php.SOA Systems Inc, 2009.

[33] Extended Web Services Standards Business Process Automation, http://www.eti.pg.gda.pl/katedry/kask/dydaktyka/Automatyzacja_procesow_biznesowych/APB2011/wWSStackAPB.pdf.
[34] Francisco Curbera, Rania Khalaf, Nirmal Mukhi, Stefan Tai, and Sanjiva Weerawarana. 2003. The next step in Web services. Commun. ACM 46, 10 (October 2003), 29-34. DOI=10.1145/944217.944234 http://doi.acm.org/10.1145/944217.944234.
[35] Freedman, R.S,"Testability of software components," Software Engineering, IEEE Transactions on , vol.17, no.6, pp.553-564.1991 doi: 10.1109/32.87281.
[36] G. Canfora and M. Di Penta. SOA: testing and self-checking. In Proceedings of the International Workshop on Web Services: Modeling and Testing (WSMaTe2006), pages 3–12. Palermo, Italy, 2006.
[37] Gartner, Inc http://www.gartner.com/technology/home.jsp.
[38] Gerardo Canfora and Massimiliano Penta. 2009. Service-Oriented Architectures Testing: A Survey. In Software Engineering, Andrea Lucia and Filomena Ferrucci (Eds.). Lecture Notes In Computer Science, Vol. 5413. Springer-Verlag, Berlin, Heidelberg.
[39] GlassFish.Metro Security Mechanism Configuration Options. Retrieved from: https://metro.java.net/2.0/guide/Security_Mechanism_Configuration_Options.html.
[40] Glenford J. Myers: The art of software testing (2.ed.). Wiley 2004, isbn 978-0-471-46912-4, pp. I-XV, 1-234.
[41] Greiler.M, Gross.H-G, Naser.K Runtime Integration and Testing for Highly Dynamic Service Oriented ICT Solutions-An Industry Challenges Report.
[42] Guilan Dai, Xiaoying Bai, Chongchong Zhao, "A Framework for Model Checking Web Service Compositions Based on BPEL4WS," icebe, pp.165-172, IEEE International Conference on e-Business Engineering (ICEBE'07), 2007.
[43] Harris. T, SOA Test Methodology,Torry Harris Business Solutions. Retrieved from: http://www.thbs.com/white_papers.html,2007.
[44] Heather Kreger. Web Services Conceptual Architecture (WSCA 1.0). 2001. Retrieved from: http://www.cs.uoi.gr/~pitoura/courses/ds04_gr/webt.pdf.
[45] Hong Zhu, "A Framework for Service-Oriented Testing of Web Services, " compsac, vol. 2, pp.145-150, 30th Annual International Computer Software and Applications Conference (COMPSAC'06), 2006 Oxford Booker University .Oxford.UK, 2008.
[46] HP Service Test :http://www8.hp.com/us/en/software-solutions/software.html?compURI=1173796#.UPrnKCfZbuY.
http://www.zapthink.com/2005/08/24/what-belongs-in-a-service-contract/.
[47] IBM Corporation, Web services overview. 2005. Retrieved from: http://publib.boulder.ibm.com/infocenter/rtnlhelp/v6r0m0/index.jsp?topic=%2Fcom.ibm.etools.webservice.doc%2Fconcepts%2Fcws.html.
[48] Inçki, K.; Ari, I.; Sozer, H., "A Survey of Software Testing in the Cloud," Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on , vol., no., pp.18,23, 20-22 June 2012 doi: 10.1109/SERE-C.2012.32.
[49] J.Gao. Component Testability and Component Testing Challenges. In International Workshop on Component-Based Software Engineering. 2000.
[50] jBlitz:http://www.clanproductions.com/jblitz/.
[51] Johannes Ryser, Stefan Berner, and Martin Glinz. 1999. On the State of the Art in Requirements-Based Validation and Test Of software. Technical Report. University of Zurich.
[52] Jones, S, "Toward an acceptable definition of service [service-oriented architecture]," Software, IEEE , vol.22, no.3, pp. 87- 93, 2005 doi:10.1109/MS.2005.80.
[53] José García-Fanjul, Marcos Palacios-Gutiérrez, Javier Tuya-González, and Claudio de la Riva-Alvarez: Methods for testing Web Service Compositions, Experiences and Advances in

Software Quality, CEPIS, Volume: 2009, Issue V, 2009.
[54] JUnit:http://www.junit.org.
[55] K. ZieliĔski, T. Szmuc: Software engineering: evolution and emerging technologies, Amsterdam.IOS Press, (2005).
[56] K.Adamopoulos, M. Harman, and R. M. Hierons, "How to overcome the equivalent mutant problem and achieve tailored selective mutation using co-evolution, " in Genetic and Evolutionary Computation Conference (GECCO), 2004, pp. 1338–1349.
[57] Kajko-Mattsson, M.; Lewis, G.A.; Smith, D.B.; , "A Framework for Roles for Development, Evolution and Maintenance of SOA-Based Systems," Systems Development in SOA Environments, 2007. SDSOA '07: ICSE Workshops 2007. International Workshop on , vol., no., pp.7-7, 20-26 May 2007 doi: 10.1109/SDSOA.2007.1.
[58] L. Bass, P. Clements, and R. Kazman. Software Architecture in Practice. Addison Wesley, Reading, Mass., 1998.
[59] Lennon, R. 2005. Optimisation of service provision for composite web services. In Companion To the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (San Diego, CA, USA, October 16–20, 2005). OOPSLA '05. ACM, New York, NY, 216- 17.DOI=http://doi.acm.org/10.1145/1094855.1094942.
[60] LISA:http://www.itko.com/.
[61] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009);The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.
[62] Michael Papazoglou. Web Services: Principles and Technology. PrenticeHall, 1 edition, September 2007.
[63] Microsoft Application Architecture Guide, 2nd Edition. Retrieved from: http://msdn.microsoft.com/en-us/library/ee658098.aspx#WhatIsSoftwareArchitecture.
[64] Mike P. Papazoglou, Willem-Jan van den Heuvel: Service oriented architectures: approaches, technologies and research issues. VLDB J. 16(3): 389-415 (2007).
[65] Mousavi,M.,Equivalence Class Testing, Eindhoven University of Technology,The Netherlands , pp,2013,retrieved from :http://www.win.tue.nl/~mousavi/2IW65/2.pdf.
[66] Nguyen, C.D., Marchetto, A., Tonella, P., "Challenges in Audit Testing of Web Services," Software Testing, Verification and Validation Workshops (ICSTW), 2011.
[67] Offutt, Xu: Generating Test Cases for Web Services Using Data Perturbation. Fairfax, VA. TAV-WEB Proceedings/ACM SIGSOFT SEN P1 Volume 29 Number 5, 2004.
[68] Oracle Corporation. HTTP Binding Component User's Guide. 2010.Retrieved from: http://docs.oracle.com/cd/E19182-01/821-0830/gggsrv/index.html.
[69] Oracle, April 2012, The Java EE 6 Tutorial. Retrieved from: http://docs.oracle.com/javaee/6/tutorial/doc/giqsx.html.
[70] P. G. Frankl and O. Iakounenko, "Further Empirical Studies of test Effectiveness, " Proc. 6th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Orlando (FL, USA), pp. 153-162, November 1-5, 1998.
[71] P.Taranti, C.J.P.D. Lucena, and R. Choren, "An Industry Use Case: Testing SOA Systems with MAS Simulators," in Proc. MALLOW, 2009.
[72] Papazoglou, M., 2005."Extending the service-oriented architecture,"Open Access publications from Tilburg University urn:nbn:nl:ui:12-3969166, Tilburg University.
[73] Parasoft SOAtest:http://parasoft.com.
[74] Perry, William E., Effective Methods for Software Testing, Wiley- QED Information Sciences, Inc., John Wiley & Sons, Inc., New York, NY, 1995,ISBN #0-471-06097-principle,.http://trese.cs.utwente.nl/taosad/separation_of_concerns.htm.
[75] PushToTest TESTMAKER: http://www.pushtotest.com.
[76] Qusay H. Mahmoud, Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI) , Oracle, April 2005.Retrieved from

:http://java.sun.com/developer/technicalArticles/WebServices/soa/.
[77] R.Jeevarathinam and A.S. Thanamani, "Test Case Generation using Mutation Operators and Fault Classification," presented at CoRR, 2010.
[78] Raimund Kirner and Susanne Kandl. Test Coverage Analysis and Preservation for Requirements-Based Testing of Safety-Critical Systems. 2008. Retrieved from:http://ercim-news.ercim.org/content/view/456/699/.
[79] Rakesh Kumar, Deepali Gupta, Metrics and Heuristics in Software Engineering, GJCST (2010) Volume 10 Issue 15: 23-26. Category: D.2.8, D.4.8, I.2.8
[80] ROCHA, Camila Ribeiro and MARTINS, Eliane. A method for model based test harness generation for component testing. J. Braz. Comp. Soc. [online]. 2008, vol.14, n.1, pp. 7-23. ISSN 0104-6500. http://dx.doi.org/10.1007/BF03192549.
[81] S. Paydar and M. Kahani, "An Agent-Based Framework for Automated Testing of Web-Based Systems,"Journal of Software Engineering and Applications, Vol. 4 No. 2, 2011, pp. 86-94. doi: 10.4236/jsea.2011.42010.
[82] Schahram Dustdar, Stephan Haslinger, Object-Oriented and Internet-Based Technologies, Chapter Title: Testing of Service-Oriented Architectures—A Practical Approach , PP 55–65, 2004.
[83] Schieferdecker, Stepien: Automated Testing of XML/SOAP based Web Services.Retrieved from:: http://www.site.uottawa.ca/~bernard/TestingWebServices.pdf.
[84] Schmelzer,R. What Belongs in a Service Contract. 2005. Retrieved from:
[85] Seely,S.,Microsoft. Understanding WS-Security. 2002.Retrieved from:http://msdn.microsoft.com/en-us/library/ms977327.aspx#understw_topic3.
[86] Service-Oriented Architecture, Wikipedia. Retrieved from: http://en.wikipedia.org/wiki/Service-oriented_architecture,2010.
[87] Services Oriented Architecture (SOA) Infrastructure Market Shares, Market Strategy, and Market Forecasts.Retrieved from: ftp://ftp.software.ibm.com/software/soa/pdf/Service_Oriented_Architecture_SOA_Infrastructure_all.pdf.
[88] Sharma, A.; Hellmann, T.D.; Maurer, F., "Testing of web services - A systematic mapping," Services (SERVICES), 2012 IEEE Eighth World Congress on , vol., no., pp.346,352, 24-29 June 2012 doi: 10.1109/SERVICES.2012.21.
[89] SOA Alliance, Group of SOA Practitioners, SOA Blueprint– Reference Architecture V1.1, , Available from:http://www.soablueprint.com/whitepapers/SOAReferenceArchitectureReformatted.pdf, 2006.
[90] SOA fundamentals in a nutshell: Prepare to become an IBM Certified SOA Associate: Mohamed I. Mabrouk.
[91] SoapUI-WebService:Testing: http://www.soapui.org.
[92] Stefan Jungmayr: Improving testability of object-oriented systems.ISBN 3-89825-781-9. Retrieved from: http://www.dissertation.de/index.php3?active_document=/FDP/sj929.pdf.
[93] Tekinerdogan,B .Separation of concerns. Retrieved from:
[94] Text classification and Naive Bayes, Cambridge University Pres. 2008.Retrieved from: http://nlp.stanford.edu/IR-book/html/htmledition/text-classification-and-naive-bayes-1.html.
[95] Tsai, W. T., Gao, J., Wei, X., and Chen, Y. 2006. Testability of Software in Service-Oriented Architecture. In Proceedings of the 30th Annual international Computer Software and Applications Conference, Volume 02 (September 17–21, 2006). COMPSAC. IEEE Computer Society, Washington, DC, USA.
[96] Tsai, W.T.; Xinyu Zhou; Yinong Chen; Xiaoying Bai; , "On Testing and Evaluating Service-Oriented Software," Computer , vol.41, no.8, pp.40-46, Aug. 2008 doi: 10.1109/MC.2008.304.

[97] Using machine learning to refine Category-Partition test specifications and test suites Original Research Article Information and Software Technology, Volume 51, Issue 11, November 2009, Pages 1551-1564 Lionel C. Briand, Yvan Labiche, Zaheer Bawar and Nadia Traldi Spido.
[98] Valecha ,G. ,Testability:Test Before Testing!, CodeProject, 2011. Retrieved from: http://www.codeproject.com/Articles/275631/Testability-Test-before-Testing.
[99] Vasilios S. Lazarou, Spyridon K. Gardikiotis and Nicos Malevris (2008). Agent Systems in Software Engineering, Tools in Artificial Intelligence, Paula Fritzsche (Ed.), ISBN: 978-953-7619-03-9, InTech, Austria.Retrieved from: http://sciyo.com/articles/show/title/agent_systems_in_software_engineering.
[100] Verification for Timed Systems: Applications to Web Services. Journal of Software,2012, 7 (6), pp.1338-1350.
[101] Vmware, vFabric 5 Documentation Center, Data Types, Retrieved from: http://pubs.vmware.com/vfabric5/index.jsp?topic=/com.vmware.vfabric.sqlfire.1.0/reference/language_ref/ref-data-types.html.
[102] Voas, J.M.; Miller, K.W., "Improving the software development process using testability research," Software Reliability Engineering, 1992. Proceedings., Third International Symposium on , vol., no., pp.114,121, 7-10 Oct 1992 .doi: 10.1109/ISSRE.1992.285852.
[103] W. T. Tsai et al. Scenario-based web service testing with distributed agents. IEICE Transaction on Information and System, E86-D(10):2130–2144, 2003.
[104] Wang, Hongbing, et al. "Web services: problems and future directions." Web Semantics: Science, Services and Agents on the World Wide Web 1.3 (2004): 309-320.
[105] Web Services Addressing 1.0 - Core, M. Gudgin, M. Hadley, and T. Rogers, Editors. World Wide Web Consortium, 9 May 2006. Available at http://www.w3.org/TR/ws-addr-core.
[106] Web Services Architecture, W3C Working Draft .2002.(Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University).Retrieved from: http://www.w3.org/TR/2002/WD-ws-arch-20021114/.
[107] Website Load Test - JBlitz Professional 5.1, Clan Productions Limited, Retrieved from:http://www.clanproductions.com/jblitz/,2010.
[108] Wei-Tek Tsai, Yinong Chen, and Ray Paul. 2005. Specification-Based Verification and Validation of Web Services and Service-Oriented Operating Systems. In Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS '05). IEEE Computer Society, Washington, DC, USA.
[109] What is TTCN-3?, ETSI CTI, available from: http://www.ttcn-3.org/WhatisT3.htm,2009.
[110] Wieland, Matthias, et al. "Institut für Architektur von Anwendungssystemen."Retrieved from: ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/TR-2008-09/TR-2008-09.pdf.
[111] Wirtz, Guido. "Distributed Systems Group." (2009).Retrieved from: https://opus4mig.kobv.de/opus4-bamberg/files/249/BBWIAI87Schwalbfinal2.pdf
[112] Ws-soa granularity, 2012. Retrieved from: http://www.ibm.com/developerworks/webservices/library/ws-soa-granularity/#resources.
[113] Xiang Li, Jinpeng Huai, Xudong Liu, Jin Zeng, Zicheng Huang, "SOArMetrics: A Toolkit for Testing and Evaluating SOA Middleware, " services, pp.163-170, 2009 Congress on Services I, 2009.
[114] Xiaoying Bai; Dezheng Xu; Guilan Dai; Wei-Tek Tsai; Yinong Chen; , "Dynamic Reconfigurable Testing of Service-Oriented Architecture, " Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International, vol.1, no., pp.368-378, 24-27 July 2007
doi: 10.1109/COMPSAC.2007.106.
[115] Xiaoying Bai; Guilan Dai; Dezheng Xu; Wei-Tek Tsai; , "A multi-agent based framework for collaborative testing on Web services, " Software Technologies for Future Embedded and

Ubiquitous Systems, 2006 and the 2006 Second International Workshop on Collaborative Computing, Integration, and Assurance. SEUS 2006/WCCIA 2006. The Fourth IEEE Workshop on , vol., no., pp.6 pp., 27-28 April 2006 doi: 10.1109/SEUS-WCCIA.2006.7.

[116] XSD restriction/Fascets, W3Schools.com. Retrieved from:http://www.w3schools.com/schema_facets.asp.

[117] Y.Prasanth , V.Sarika , D.Santhosh Anuhya , Y.Vineela , A. Ajay Babu . "Framework for Testing Web Services Through SOA (Service Oriented Architecture)". International Journal of Engineering Trends and Technology (IJETT). V3(2):103-109 Mar-Apr 2012. ISSN:2231-5381

[118] Ying-Dar Lina, Chi-Heng Choua, Yuan-Cheng Lai, Tse-Yau Huang, Simon Chung, Jui-Tsun Hunga and Frank C. Line, "Test coverage optimization for large code problems," Journal of Systems and Software archive, Vol. 85,No. 1,pp.16–27, 2011.

[119] Yoon, H., Ji, E., and Choi, B. 2008. Building test steps for SOA service orchestration in web service testing tools. In Proceedings of the 2nd international Conference on Ubiquitous information Management and Communication (Suwon, Korea, January 31 - February 01, 2008). ICUIMC '08. ACM, New York.

[120] Youngkon Lee, "2-Layered SOA Test Framework Based on Event-Simulating Proxy, " ncm, pp.1479-1484, 2009 Fifth International Joint Conference on INC, IMS and IDC, 2009.

[121] Yu Qi, David Kung and Eric Wong, ``An Agent-based Data-Flow Testing Approach for Web Applications," Journal of Information and Software Technology, July 2006.

[122] Yue Jia; Harman, M.; "An Analysis and Survey of the Development of Mutation Testing " Software Engineering, IEEE Transactions on , vol.37, no.5, pp.649-678, Sept.-Oct. 2011.

[123] Yunus, Mallal :SOA Testing using Black, White and Gray Box Techniques. Crosscheck Networks.

[124] Yunus,M., Mallal,R., "Watch your SOA Testing Blind Spots", Crosscheck Networks. Retrieved from:http://www.softwaremag.com/pdfs/whitepapers/Crosscheck_wp2.pdf?CFID=30679347&CFTOKEN=35280601.