2010

Multi-tier framework for management of web services' quality

Abdelghanni Benharref

M. Adel Serhani

Mohamed Salem
salem@uow.edu.au

Rachida Dssouli

Follow this and additional works at: <https://ro.uow.edu.au/dubaipapers>

Recommended Citation

Benharref, Abdelghanni; Serhani, M. Adel; Salem, Mohamed; and Dssouli, Rachida: Multi-tier framework for management of web services' quality 2010.
<https://ro.uow.edu.au/dubaipapers/360>

*Multi-tier Framework for **Management** of **Web** **Services**' Quality*

*Abdelghani Benharref, **Mohamed Adel Serhani, ***Mohamed Salem, *Rachida Dssouli,

*Department of Electrical and Computer Engineering, Concordia University, 1515 Ste-Catherine West, Montreal, Quebec, H3G 2W1
Canada

Tel.: +1 514 848 2424 Extension 4162

Fax: +1 514 848 3171

Email: {dssouli, abdel}@ece.concordia.ca

**College of Information Technology, P.O.Box 17555

United Arab Emirates University, Al-Ain

United Arab Emirates

Tel.: +971 37 31 55 72

Fax: +971 37 67 20 18

Email: serhanim@uaeu.ac.ae

***College of Information Technology, P.O. Box 20183

University of Wollongong, Dubai

United Arab Emirates

Tel.: +971 4 367 2465

Fax: +971 4 367 2754

Email: mohamedsalem@uowdubai.ac.ae

*Multi-tier Framework for **Management** of **Web Services**' Quality*

Keywords: Web Services' management, Quality of Web Services, Web Services monitoring, Quality of Web Services' specification, quality of Web Services' verification, Web Services-aware selection, Quality of Web Services-publication, composite Web Services

ABSTRACT

Web Services are new breed of applications that endorsed large support from main vendors from industry as well as academia. As the Web Services paradigm becomes more mature, its management is crucial to its adoption and success. Existing approaches are often limited to the platforms under which management features are provided. In this chapter, we propose an approach to provide a unique central console for management of both functional and non-functional aspects of Web Services. In fact, we aim at the development of a framework to provide management features to providers and clients by supporting management activities all along the lifecycle. The framework allows/forces providers to consider management activities while developing their Web Services. It allows clients to select appropriate Web Services using different criteria (name, quality). Clients make also use of the framework to check if the Web Services, they are actually using or planning to use, are behaving correctly. We evaluate the Web Services management features of our framework using a composite Web Service.

INTRODUCTION

Web Services standard is a recent paradigm of emerging web components. It combines a set of technologies, protocols, and languages to allow automatic communication between Web applications through the Internet. A Web Service is any application that exposes its functionalities through an interface description and makes it publicly available for use by other programs. **Web Services** can be accessed using different protocols, different component models, and running on different operating systems. They usually use Hypertext Transfer Protocol (HTTP) (W3C, 1999) as a fundamental communication protocol, which carries exchanged messages between **Web Services** and their clients. **Web Services** use eXtensible Markup Language (XML) (W3C, 2006) based messaging as a fundamental means of data communication.

Research on **Web Services** has focused more on interfacing issues, i.e., Simple Object Access Protocol (SOAP) (W3C, 2004), Web Services Description Language (WSDL) (WSDL, 2001), and Universal Description, Discovery and Integration (UDDI) (OASIS, 2005). Until recently, considerable efforts have been conducted to address the issues of **management** of **Web Services** in **Service Oriented Architecture** (SOA).

Web Services' **management** is among the hot issues that are not yet mature. Ongoing research from academia and industry are still emerging. **Management** of **Web Services** is critical for their

success because they are being actually used in a wide range of applications, ranging from entertainment, finance, and healthcare to real-time critical applications. **Management** issues in Web Service can be divided into two dimensions: (1) **management** of **functional** aspects, namely fault management and (2) management of **non-functional** aspects such as Quality of Service (QoS). **Quality of a Web Service**, referred to as **QoWS** in this chapter, reflects the quality of a Web Service, both in terms of correctness of **functional** behaviour and level of supported QoS. A Web Service supporting **QoWS** is said to be **QoWS-aware**.

Nowadays, **management** of **Web Services** is highly platform-dependent which implies the following limitations: (1) management features are usually available to **Web Services**' providers but often not to other partners (clients, third parties), (2) management solutions are usually restricted to only one management aspect, **functional** or **non-functional**, and (3) most of management solutions require considerable amount of computer and network resources to be deployed and used.

The first limitation restricts the utilization of management information to providers who are using it to assess the **QoWS** of their **Web Services**. However, other entities involved in SOA industry might need to use this information as well. Clients can use this information during discovery and selection of **Web Services** so they can figure out those with desirable **QoWS**. Moreover, many providers are likely to offer **Web Services** providing similar functionalities but with quite different **QoWS**. In such a competitive market, attraction and loyalty of clients are primarily based on high standards of provided **QoWS**.

In SOA, a significant amount of work is taking place to allow both **Web Services**' providers and their clients to define and concisely use **QoWS** during publication, discovery, and invocation of **Web Services**. For example, to select from a set of potential **Web Services**, the one which is mostly available, has a low response time, and/or an acceptable utilization fee.

This chapter presents our approach for **management** of **Web Services**. This approach provides a unique central environment for management of both **functional** and **non-functional** aspects of **Web Services**. In fact, we aim at the development of a framework to provide management features to **Web Services** providers and clients by supporting management activities all along the lifecycle of a Web Service, from specification to invocation. The framework allows/forces providers to consider management activities while developing their **Web Services**. In fact, the provider should concisely and precisely describe **QoWS** factors during design and implementation of the Web Service. These factors will/shall be used latter by clients to select appropriate **Web Services** during the discovery and selection operations. Clients make also use of the framework to check if the **Web Services**, they are actually using or planning to use, are behaving correctly in terms of **functional** and **non-functional** facets.

The concepts presented all along this chapter will be illustrated in details through a case study. A Web Service example will be used to show how different phases of the Web Service development life cycle must be conducted while promoting good practices for advanced management activities. At each phase, information and documents required by the management framework are produced and their impact on management is thoroughly discussed.

This book chapter is organized as follows: the next section provides background information required for non-expert readers to follow the flow of ideas in following sections. The following section discusses related work in **management** of **Web Services** and their limitations. A composite Web Service used to illustrate our management framework is introduced. The

subsequent section details, in a step-by-step tactic, how different management activities proposed by our framework can be conducted at each phase during the development of a Web Service. We show then promising experimental results while using the framework to manage the Web Service introduced hereafter. We close the chapter by presenting conclusions and insights for ongoing and future work.

BACKGROUND

Web Services are a new variant of web applications. It is a new paradigm which allows different applications to communicate automatically with each other over the Internet. They are self-contained, self-describing, modular applications that can be published, located and invoked across the Internet (Wahli, 2002). The endeavor of this new paradigm is to allow applications to be delivered over the Internet and to run across all kinds of computers and platforms.

A Web Service is any application that can be published, located, and invoked through the Internet. Each Web Service has a Web Service Description Language document (WSDL)(W3C, 2001); which consists of an XML (W3C, 2006) document providing all required knowledge to communicate with the Web Service including its location, supported transport protocols, messages formats, list and signatures of published operations.

A Web Service can perform any kind of transactions that may range from getting a city's temperature to a more complicated transaction like for instance searching and/or building the best travel packages from specific travel agencies. The main objective of **Web Services** is to allow, at a high level of abstraction, applications to be accessible over the Internet. They can be of great use, for instance, for 3G networks operators to expose their core network functionalities to third parties (3GPP, 2003) and for digital imaging where they can provide an important benefit to the digital photography industry. The Common Picture eXchange environment (CPXe) (CPXe, 2005), a Web Service business framework, will make transfer and printing of digital images as suitable as the use of films.

SOA defines three roles (provider, requester, and registry) and three operations (publish, find, and bind). The relationship between the roles and the operations are illustrated in Figure 1. Additional information on the **Web Services** architecture can be found in (Kreger, 2001).

The starting point in **Web Services** activities is the development, deployment and publication of the Web Service by its provider. When a requestor (client) needs a specific Web Service, she/he probes the registry for a list of potential **Web Services**. The returned list contains matching records; each record contains required information to connect to the corresponding Web Service. Based on a set of criteria (location, availability ...), the requestor selects a suitable Web Service and binds to it.

Web Services can be developed either from scratch or by composition. Composition of **Web Services** is the process of aggregating a set of **Web Services** to create a more complete Web Service with a wider range of functionalities. This composition has a considerable potential of reducing development time and effort for new applications by reusing already available **Web Services**.

Figure 1. Service Oriented Architecture

Currently, there are standards or languages that help building composite **Web Services** such as: Web Services Flow Language (WSFL) (Leymann, 2001), DAML-S (Ankolekar et al., 2002), Web Services Conversation Language (WSCL) (Banerji et al., 2002), Web Services Choreography Interface (WSC) (Arkin et al., 2002), and Business Process Execution Language (BPEL) (Andrews et al., 2003). These languages make easier **Web Services'** composition process by providing concepts to represent partners and orchestrate their interactions. BPEL, which represents the merging of IBM's WSFL and Microsoft's XLANG, is gaining a lot of interest and is positioned to become the primer standard for Web Service composition.

RELATED WORK

Most works on **Web Services** focus on their development and deployment. **Management of Web Services** (W3C, 2002), and in particular fault and performance management, are not yet a well-studied area. However, some interesting works have to be cited.

Existing approaches for **management** of **Web Services** include approaches from network management and those that have been developed specifically for **Web Services**. The approaches that have been used for network management for a long time seem to be a candidate for the **management** of **Web Services**. However, their main drawbacks are due to the major differences between **Web Services** and network components, and the need for the participation of a component in its management. In fact, most network components (devices) run standardized protocols that have specific and known attributes to be managed. Manufacturers of components running proprietary/non-standard protocols and/or applications often provide their customers with specific management agents/applications or well-defined sets of APIs.

In network oriented approaches, Simple Network Management Protocol (SNMP) (Case, Fedor, Schoffstall, & Davin, 1990) is based on TCP/IP and the client/server communication mode. In this approach, an agent associated with a Management Information Base (MIB) (Perkins & McGinnis, 1997), communicates with a management station by processing *get* (report the value of an attribute) and *set* (modify the value of an attribute) messages and generating *trap* messages (unsolicited notification). Thus, SNMP management system requires a SNMP agent, a MIB, and a management station (manager).

The Common Management Information Protocol (CMIP) (ISO/IEC, 1998) fulfills in the OSI reference model protocol stack (ISO/IEC, 1989), a role similar to that of SNMP in TCP/IP. CMIP has many advantages compared to SNMP including the number of available commands and the possibility to operate over TCP/IP. However, complexity and long development time, especially CMIP over TCP/IP (CMOT) (Warrier, Besaw, LaBarre, & Handspicker, 1990), have kept its adoption pervasively.

A considerable amount of work in the [Web Services](#) community is dedicated to the determination of the requirements and the definition of specific approaches for [Web Services management](#). These approaches can be divided into two main groups: approaches based on active testing and approaches requiring the Web Service (architecture) to support management interfaces. The World Wide Web Consortium presents a set of requirements that [Web Services management](#) architectures should satisfy to provide management features (W3, 2004). This includes the definition of standard metrics, management operations, and methodologies for accessing management capabilities. The complying architectures must provide a manageable, accountable and organized environment for [Web Services](#) operations. It must support at least, resource accounting, usage auditing and tracking, performance [monitoring](#), availability, configuration, control, security auditing and administration, and service level agreements. Another approach in which the Web Service provides specific interfaces for management is presented in (Farrell & Kreger, 2002). The developer is supposed to supply commands and APIs for management operations that are invoked by the management system.

In (Casati, Shan, Dayal, & Shan, 2003), the authors classify the [management](#) of [Web Services](#) into three levels: infrastructure-level, application-level and business-level. The infrastructure-level deals with the Web Service platform while the application-level focuses on the [Web Services](#) themselves. The business-level takes into consideration the conversations between a Web Service and its client

Management approaches presented in (W3, 2004), (Farrell & Kreger, 2002), and (Casati et al., 2003) assume that the Web Service will provide management operations that one can invoke. Developers of [Web Services](#) have then to develop and deploy these operations in addition to the core business operations the Web Service is offering.

A couple of management tools to be integrated into [Web Services](#) environment are already available. Hewlett Packard's Web Service Management Engine (HP, 2007) is a collection of software components that enables some management features including the definition and the enforcement of Service Level Agreement (SLA). Parasoft (Parasoft, 2006) provides a set of tools (SOAPTest, .TEST, WebKing) to assist during the life cycle of a Web Service. These tools have to be installed and configured, thus requiring extra resources and introducing new cost for [Web Services](#) providers.

There has been a considerable amount of work on testing [Web Services](#) in the last couple of years. They can be divided into two main groups: works targeting [functional](#) aspects of [Web Services](#) and works tackling [non-functional](#). The first group is concerned with the correctness of interactions between [Web Services](#) and their clients while the second group is concerned with QoS management of [Web Services](#).

Functional Management

The majority of work on [functional](#) management is based on active testing where appropriate test cases have to be carefully generated, executed, and their results analyzed. This unavoidable phase of active testing has however practical limitations. First of all, exhaustive testing is impractical for quite large [Web Services](#). In fact, test cases can not cover all possible execution scenarios that a Web Service will have to handle while serving clients' requests. The size of test cases is bounded by the cost a Web Service's provider is willing to spend on testing activities. Usually, active testing stops whenever developers are confident that the Web Service is good enough to be put into the market.

Many recent results were published lately describing test cases generation methods for [Web Services](#); they are mainly based on static analysis of WSDL documents. Xiaoying et al. (Xiaoying, Wenli, Wei-Tek, & Yinong, 2005) presented a method for test data generation and test operation generation based on three types of dependencies: input, output, and input/output. In (Jiang et al., 2005), the authors proposed a method for test data generation in which a set of tests is randomly generated based on the WSDL document. In (ChangSup, Sungwon, In-Young, Jongmoon, & Young-Il, 2006), the authors combined both EFSM models and WSDL documents to generate test cases.

QoS Management

[QoWS](#) management includes definition of QoS attributes, QoS publication, discovery, validation, and [monitoring](#). Existing approaches for QoS management can be classified into two groups: one based on extending related technologies including WSDL and UDDI to support QoS and the other mandating independent entities to perform some or all of QoS management tasks.

In the first category, (W3C, 2003) extends SOAP header to include QoS information. WSDL is also extended to describe QoS parameters, their associated values, computation units (e.g. millisecond, request/second), etc. UDDIe, a UDDI extension, consists of extending the current UDDI data structure with QoS information (ShaikhAli, Rana, Al-Ali, & Walker, 2003). The aim of these extensions is to allow QoS-based publication and discovery of [Web Services](#).

In the second group, solutions are presented for one or more of the following QoS management operations:

- QoS attributes: the first step in QoS management is the definition of evaluation's criteria and attributes. A set of attributes have been defined, studied, and used in software engineering for a long time ((Fenton & Pfleeger, 1997), (Gray & MacDonell, 1997), (Salamon & Wallace, 1994)).
- QoS publication and discovery ((Ran, 2003), (Serhani, Dssouli, Hafid, & Sahraoui, 2005), (Kalepu, Krishnaswamy, & Loke, 2004)): this operation allows providers to

include QoS information in WSDL. This information is then used by requestors when selecting the appropriate Web Service in terms of **functional** and QoS requirements.

- QoS verification ((Serhani et al., 2005), (Kalepu et al., 2004), (Tsai, Paul, Cao, Yu, & Saimi, 2003)): this operation allows the provider to certify that the QoS claimed by the Web Service is accurate.
- QoS negotiation (Serhani et al., 2005): if the available published QoS requirements do not satisfy client's needs, negotiation operations and strategies can be followed to reach an agreement on different QoS attributes.
- QoS **monitoring** ((Ho, Loucks, & Singh, 1998), (Yuming, Chen-Khong, & Chi-Chung, 2000), (Schmietendorf, Dumke, & Reitz, 2004), (Benharref, Glitho, & Dssouli, 2005), (Benharref, Dssouli, Glitho, & Serhani, 2006)): performs **monitoring** of **Web Services** during interactions with clients to assess if the QoS attributes agreed upon in previous points are delivered.

Discussion

All the solutions presented above fit in one or more of the following categories:

1. Platform-dependent
2. Assume that a Web Service will participate in its management by providing specific interfaces (e.g. W3C architecture),
3. Are based on active testers.

The usage of platform-dependent management approaches is restricted to the targeted platform. When management features are embedded to the hosting platform, they are only available to the provider and cannot be used by clients or third party certification entities. A client might need management information for two tasks: (1) during discovery and selection to select the appropriate Web Service, and (2) during invocation to assess the quality of the interactions. The client must rely on management information made available by the Web Service provider and has no mean of verifying it. Moreover, information used in assessing the behavior is taken from one location: at the provider's side. There are many situations, in composite Web Service for example, where this information should be gathered from different sources and locations.

The **Web Services** architecture becomes more complex if it has to support management features in addition to its basic functions. The performance of the Web Service and its hosting platform is also degraded due to these additional features. Moreover, developers of **Web Services** have to implement also the needed interfaces and APIs to support management. Since these features will be used somehow sporadically, the return on investment of their development and deployment might be relatively low.

Once a Web Service is interacting with clients, active testing cannot be used to monitor, in real time, the correctness of interactions. Moreover, application of generated test cases consumes resources and may disturb the Web Service.

Since **management** of **Web Services** is somehow at its earlier stages, related work usually concentrates more on provision of management features without evaluating the overhead they

generate. In order to select the appropriate management approach, a potential user must be able to evaluate it in terms of usefulness and associated cost.

Furthermore, most of existing work on **management** of **Web Services** doesn't tackle management issues at the earlier phase of their development. However, management features need to be addressed as early as possible in the development process especially during the design and implementation phases. For example design for manageability will describe manageability scope and functions. Moreover, it will expose a Web Service as a manageable entity providing some of the following capabilities (Farrell & Kreger, 2002): operations, events, interfaces, status, configuration, and metrics that can be used for managing and controlling **Web Services**.

To solve some of the limitations of related work cited above, this chapter presents a novel framework for **management** of **Web Services**. This framework considers **QoWS** management issues from earlier phases of the development life cycle of a Web Service. These issues are specified during specification and design, verified and certified before deployment, published with the WSDL document, used during discovery and selection, and passively monitored during invocation of the Web Service.

To illustrate the applicability of our approach for **management** of **Web Services**, we will be using a case study all along the chapter's sections to demonstrate how each **management** task, at each development phase, can be achieved. Introduction to this case study and its utilization context are given in the next section.

CASE STUDY

For the end of year meetings, a general manager has to meet with managers from different departments (e.g. Sales, R&D...). Managers are located in different locations and, because of their busy timetables, they cannot meet in a single location. A practical alternative is to conduct these meetings in a series of teleconferences. Only managers are concerned and only those of them who are in their offices can join a conference. This is implied by security issues since confidential information will be exchanged during meetings and communication between different locations is secured (VPN for example). At the end of each meeting, meetings' reports must be printed and distributed among all participants.

The manager decides to use available conferencing **Web Services**. Such **Web Services** should allow creation of conferences, and the addition and removal of participants to conferences depending on their profiles and physical locations. At the end of each meeting, the Web Service should be able to submit produced reports for printing and deliveries.

The general manager is highly concerned with the environment in which meetings will be carried out using **Web Services**. A thorough **QoWS**-based discovery and selection operations had lead to the utilization of "Conferencing Web Service" (CWS), a **QoWS**-aware composite Web Service, which performs all of the required tasks. The manager decides to make use of the **monitoring** feature of the management framework to assess the **QoWS** of the CWS.

To perform all these tasks, the CWS is a composition of the following basic **Web Services** (Figure 2):

Figure 2. Composite/basic Web Services

- Presence: this Web Service contains information on managers' profiles (name, address, location, status, position, availability).
- Sensors: this Web Service detects the physical location of managers.
- Call Control: this Web Service creates and manages multiparty conferences (initiates conferences, adds/removes participants, and ends conferences).
- Printing: at some points during conferences or later on, managers may want to print documents (meeting reports...). The printing Web Service will print these documents and keeps them for shipping.
- Shipping: documents printed during and after the conference should be distributed among participants at different locations. The CWS informs the shipping Web Service of the location of the documents to be shipped and their final destinations.

WEB SERVICES' MANAGEMENT FRAMEWORK

For a better exploitation of Web Services, a Web Services lifecycle is supposed to integrate features such as QoWS factors' precise definition, QoWS specification, QoWS-based discovery and selection, and QoWS monitoring. This implies that these features need to be addressed in earlier phases through Web Services development process especially during the design phase, then ultimately in the implementation phase, and possibly during selection and invocation of the Web Service. QoWS management information, for example, is first specified then published to be latter on discovered by clients using QoWS-aware discovery. Our approach is to investigate possibilities to augment the development process of Web Services with the above important features.

In the subsequent sections, we will describe how management of QoWS should be supported during development, publication and deployment, discovery, selection, and invocation.

During Development of the Web Service: Behavior's Specification

The first step in a QoWS-aware development and utilization of Web Services is the selection and concise definition of factors that will characterize the quality of a Web Service. As cited above, these factors are divided into functional and non-functional.

Functional Aspects

Definition of functional attributes specifies the correct functional behavior of the Web Service for each invocation of each published operation. This covers the content of invocations, their responses, and their sequence.

Two main ways for functional behavior's description have been studied in the literature: formal models and knowledge bases/expert systems (Vijayananda & Raja, 1994). Formal models have many advantages over expert systems. First of all, expert systems rely on human expertise and are more appropriate for systems that have been encountered previously. Second, formal models can be useful for automatic generation of source code and executable test cases. For these reasons, Finite State Machines (FSM) (Dssouli, Saleh, Aboulhamid, En-Nouaary, & Bourhfir, 1999), a widely known formal model, will be used to specify expected functional behaviors of Web Services in this chapter.

A Finite State Machine M is defined as a tuple $(S, S_0, X, Y, D_S, \delta, \lambda)$, where:

- S is a set of states,
- $S_0 \in S$ is the initial state,
- X is a finite set of inputs,
- Y is a finite set of outputs,
- $D \subseteq S \times X$ is the specification domain,
- $\delta: DS \rightarrow S$ is the transfer function, and
- $\lambda: DS \rightarrow Y$ is the output function

The machine starts at S_0 . Whenever an input is received, λ computes the corresponding output and δ determines the corresponding next state(s).

An FSM can be represented by an XML document as illustrated in Figure 3, which gives a partial overview of the FSM machine of the CWS. The root of the document (*fsm*) has an attribute (*name*) and a set of children which represents states. The name is a textual description of the Web Service. Each child has a name, the attribute (*initial*), and a set of *transitions*. The name is a textual description of the state while the attribute "initial", if set to YES, indicates that this is the initial state of the machine. A transition has four attributes: *ID*, *input*, *output*, and *next*. The first attribute is a textual description of the transition; the second attribute identifies the event that triggers this transition if the machine is in the associated state; the third attribute is the

output generated when firing that transition; and the last attribute specifies the state that the machine will reach after firing the transition.

```
<fsm name="Conferencing Web Service ">
  <state name="Init" initial="YES">
    <transition ID="t1"    input="Config_Valid" output="True" next="Ready"/>
    <transition ID="t2"    input="Config_Invalid" output="False" next="Init"/>
  </state>
  <state name="Ready" initial="NO">
    <transition ID="t3"    input="CreateConf_Valid" output="True"
                      next="ConfCreated"/>
  </state>
  <state name="ConfCreated" initial="NO">
    <transition ID="t4"    input="AddUser" output="True" next="ConfCreated"/>
  </state>
  ...
</fsm>
```

Figure 3. XML representation of an FSM machine

Non-functional Aspects: QoS

QoS consists of a set of factors or attributes such as response time, reliability, availability, accessibility, etc. Information on QoS attributes can be specified in many different ways. It can be described in a separate document, embedded within the description of **functional** behavior, or as an extension to WSDL document. However, to allow **QoWS**-aware discovery and selection of **Web Services**, **QoWS** attributes should be available within the WSDL document. The client indicates preferences in terms of **QoWS** when probing the registry. The registry returns then a list of available **Web Services** providing required operations with requested **QoWS**.

The first step in extending SOA with QoS is the definition of QoS attributes. In this chapter, we will focus on the following attributes:

- **Processing Time (PT)**: this is a measure of the time a Web Service takes between the time it gets a request and the moment it sends back the corresponding response. PT is computed at the Web Service's provider side.
- **Maximum Processing Time (MxPT)**: this is the maximum time the Web Service should take to respond to a request.
- **Minimum Processing Time (MnPT)**: this is the minimum time the Web Service should take before responding to a request. Unlike PT which is a dynamically computed attribute, MnPT and MxPT are statically defined and: $MnPT \leq PT \leq MxPT$.

- Response Time (RT): it consists of the time needed between issuing a request and getting its response. It is measured at the client's side to include the propagation time of requests and responses.
- Maximum Response Time (MxRT): this is the maximum accepted time, for the client, between issuing a request and getting its response.
- Minimum Response Time (MnRT): this is the minimum time, for the client, between issuing a request and getting its response. This attribute is unlikely to be used since the client is usually more interested in MxRT. For the client: $RT \leq MxRT$ must always be satisfied.
- Availability: this is a probability measure that indicates how much the Web Service is available for use by clients. It can also consist of the percentage of time that the Web Service is operating.
- Service Charge (SC): it defines the cost a client will be charged for the Web Service's utilization. SC can be estimated by operation, type of requests, period of utilization, session, or by volume of processed data.
- Reputation: this is a measure of [Web Services](#)' credibility. It depends basically on previous end users' experiences while using the Web Service. Different users may have different opinions on the same Web Service. The reputation value can be given by the average ranking given to the Web Service by several users.

MnPT, MxPT, availability, and SC are related to profiles of users of the Web Service. This profiling is based on the type of subscriptions of clients and/or the [QoS](#) they are willing to pay for. For example, a gold-subscribed user must be served quicker ($MnRT = 0$) than a bronze-subscribed user ($MnRT > 1ms$).

Figure 4 illustrates embedded QoS attributes in the definition of the operation tag within the WSDL document of the CWS.

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:portType name="ConferenceService">
  <wsdl:operation name="addUser" parameterOrder="userAdresse callID">
    <wsdl:input message="intf:addUserRequest" name="addUserRequest"/>
    <wsdl:output message="intf:addUserResponse" name="addUserResponse"/>
    <Profile name="GOLD">
      MnPT = NULL
      MxPT = 10ms
      SC= "$10"
    </Profile>
    <Profile name="SILVER">
      MnPT = 10ms
      MxPT = 30ms
      SC= "$5"
    </Profile>
  </wsdl:operation>
</wsdl:portType>

```

Figure 4. QoWS in WSDL document

Before Deployment and Publication: QoWS Verification and Certification

Once a Web Service is developed, it must be tested to verify whether it is correct with regards to the behavior's **specification** document produced during preceding development phases. The management framework has features that a Web Service's developer can use to verify and certify the Web Service's behavior. This certification information is then published with the WSDL description of the Web service so potential clients will use it.

Verification and certification procedures enable providers to evaluate **QoWS** of their **Web Services** prior to the publication. Our approach consists of a two-phase verification and certification technique, which is conducted by a verifier Web Service and a certifier Web Service (Figure 7). The first phase consists of verifying the WSDL document including the **QoWS** parameters description. The second phase consists of applying a measurement technique to compute the **QoWS** metrics stated in the Web Service interface and compares their values to those claimed in the WSDL document. This is used to verify the conformity of a Web Service to its description from a **QoWS** point of view (**QoWS** testing). Therefore, a set of **QoWS** test cases are defined and used as input to **QoWS** verification. The configuration and generation of these test cases is described in details in (Serhani et al., 2005). Once the Web Service passes the verification tests, the certifier issues a conformance certificate to certify that **QoWS** claims are

valid. This certificate will be considered as a key differentiator between **Web Services** offering similar functionalities. The verifier and certifier perform the following tasks:

- It asks for information about the provider and its Web Service (Servers' resources capacity, connections used, Network information, etc.).
- It checks the WSDL files of the target **Web Services** (location, interface and implementation description)
- It makes sure that all published operations are available.
- It verifies the **QoWS** described in WSDL. The **QoWS** verifier can initiate, if necessary, additional tests to validate other information provided in the WSDL document. This information concerns **QoWS** attributes classification (definition, computation logic, upper and lower bounds).
- It stores the verification report in specific-purpose database.

During Discovery and Selection: **QoWS-based Discovery and Selection**

In standard SOA, the find operation is based on the name of the **Web Service**. A client is likely to get a list of **Web Services** following a basic find operation issued to a UDDI registry. Alternatively, an intelligent find operation must consider, in addition to the name, **QoWS** information verified and certified in the previous phase, so the returned list of **Web Services** is somehow short and concise.

Our framework defines how an intelligent discovery operation should look like and how it can be used. Discovering a Web Service will be based on its functionalities as well as its **QoWS**. We automated the processes of publication and discovery of **Web Services** based on **QoWS** using a supporting application (Figure 5), which allows the following tasks:

QoWS Validation before publication

QoWS-based discovery of Web Services

Figure 5. UDDIe client's application for publication and discovery driven QoWS

Publication: In order to publish their **Web Services** using our application, providers should supply the Web Service name, description, and the location of its WSDL document. This document is then parsed to validate its content and to display the list of **QoWS** information. The validation process verifies the correctness of published operations in addition to the **QoWS** information. The provider can add/remove/modify **QoWS** attributes before publication. At this stage, the WSDL document is validated and the provider publishes the **QoWS**-enabled WSDL document.

Discovery: the application allows the user to query the registry while specifying the **Web Service** name and the set of required **QoWS** attributes and their related values. The list of **Web Services**' descriptions that fulfill the client's requirements is displayed via the application interface. The frame contains corresponding **Web Services** and their associated **QoWS** information.

During Invocation: QoWS Monitoring

During interactions between a Web Service and its client, it might be necessary to assess if the Web Service is behaving as initially specified and claimed in previous subsections. This

assessment will require a continuous online **monitoring** of interactions between the **Web Service** and its client.

Online **monitoring** of **Web Services** requires passive observers ((Benharref et al., 2005), (Benharref et al., 2006)). A passive observer receives a copy of all exchanged messages between a Web Service and its client and checks their validity. Passive observation of systems modeled as FSM is usually performed in two steps (Lee, Netravali, Sabnani, Sugla, & John, 1997):

1. **Passive homing** (or state recognition): in this step, the observer is brought to a state equivalent to the one that the Web Service might be in. If no such state is found, a fault is immediately reported. The set of messages leading to this state is known as the homing sequence. This first step is required if observation starts while the Web Service and its client already exchanged some messages. These messages will not be available to the observer but the latter can figure out a homing sequence to determine the appropriate state.
2. **Fault detection**: starting from the state identified in the previous step, the observer checks the observed behavior against the system's specification. If an observed event is not expected then a fault is immediately reported.

The observation in distributed architectures requires the selection of a number of observers and their best locations (where to get copies of exchanged messages). The number and location of the points of observation affect significantly the detection capabilities of the observation architectures. For example, if the observed Web Service is a composite **Web Service**, it might be more interesting (in terms of misbehavior's detection) to consider a network of observers: an observer for each Web Service rather than a unique observer for the composite Web Service. In such architectures, cooperation of all observers can generate pertinent information for **Web Services** management. The consideration of a global observer (for the composite Web Service) and local observers (for composing **Web Services**) presents a framework where this cooperation can be orchestrated for the benefit of better misbehaviors' detection.

Our **Web Services management** framework offers two **monitoring** architectures as depicted in Figure 6: a mono-observer architecture (Benharref et al., 2005) and multi-observer architecture (Benharref et al., 2006).

Figure 6. Monitoring architectures

Three types of interactions are illustrated in Figure 6: **Web Services'** communication refers to the SOAP-based communication between **Web Services** and their clients. Traces' collection consists of forwarding messages exchanged between the observed Web Service and its client to local observers. Observers' communication conveys information between observers. This information is divided into three categories:

1. Configuration information: during configuration of different observers, local observers must indicate to the global observer which Web Service they are observing and where they are located. The global observer needs this information to identify observers and associate the traces it will receive to appropriate observers/**Web Services**.
2. Traces from local observers to the global observer: whenever a local observer gets a trace, it sends it to the global observer.
3. Notifications of faults: if the global observer detects a fault, it informs other local observers. In the case where a local observer detects a fault, it informs the global observer. The latter informs remaining local observers that misbehavior has been observed elsewhere and they should be aware of some specific traffic/actions.

Traces' collection mechanisms studied in (Benharref et al., 2005) showed that mobile agents present the least overhead. Whenever an entity wants to use **monitoring** architectures, it invokes a Web Service Observer (WSO) that generates a set of mobile observers and sends them to locations specified during invocation.

Except **specification** of expected behaviors of **Web Services** which has to be done by the provider, other management operations presented in previous sections are performed by invoking the verifier, the certifier, and the observer, three **Web Services** provided by the management framework. These components and their associated operations are illustrated in Figure 7:

Figure 7. Management framework components and operations

Different steps discussed above have been applied to a set of [Web Services](#) to illustrate their applicability. The next section shows an example of application to the CWS introduced earlier in this chapter.

EXPERIMENTATION AND RESULTS

As indicated earlier, CWS is a composite Web Service for management of teleconferences, printing, and shipping of teleconferences reports. Experimenting with the framework using this Web Service required implementation and deployment of a dozen of [Web Services](#).

Implementation Issues

All [Web Services](#), including the WSO, are implemented in BEA WebLogic (BEA, 2004). Mobile observers get traces using SOAP Handlers, which are available within the BEA platform. A SOAP Handler, a specific-purpose Java class, intercepts a request/response to/from a Web Service before it gets to the core Web Service or the client respectively, and can also perform operations on it. In our case, the SOAP handler sends each intercepted request or response in a User Datagram Protocol (UDP) datagram to the concerned mobile observer all along with the date at which this event occurred to allow the observer to assess QoS attributes such as response time. The mobile observer checks this trace and forwards it to the global observer.

To be able to detect lost UDP datagrams, a sequence number field is used. When a mobile observer detects a lost datagram (wrong/not expected sequence number), it suspends the misbehavior detection and re-performs the homing procedure. It restarts the detection once this procedure is achieved correctly. Since the behavior/operation of SOAP handlers within all observed [Web Services](#) is similar, a unique (generic) SOAP Handler is developed and then distributed to all providers participating in the observation.

The overhead of the management framework can be quantitatively evaluated with regards to required computer resources and generated network overhead. Both analytical analysis and experimentations showed that most of the overhead is related to the online [monitoring](#). In fact, the verification and certification operations are straightforward and usually conducted offline, that's, before the Web Service is made available to clients. Moreover, all required resources are located at the Verifier and Certifier [Web Services](#) providers. For these reasons, overhead analysis presented in upcoming sub-sections will concentrate on online [monitoring](#), especially traces' collection.

Monitoring

In addition to the observation of the CWS, the manager wants to make sure that all the steps are performed according to the agreed on contract and [QoWS](#). Fortunately, all the providers accept to participate, to some extent, in the [monitoring](#). The provider of the CWS will host all mobile observers using the Jade platform (Jade, 2007). This provider will also supply WSDL documents and FSM models of each of the basic [Web Services](#). Basic [Web Services](#)' providers will configure SOAP handlers for traces collection and forward.

The observation procedure of CWS is performed following the steps detailed below and illustrated in Figure 8. To keep the figure simple, just one Web Service handler and one Web Service client¹ are depicted in the figure.

1. The manager invokes the WSO, providing different locations of mobile observers.
2. The WSO generates a mobile agent and sends it to one of the destinations submitted during invocation in step 1.
3. Once the mobile agent gets into its destination, it clones itself as many as required to observe all [Web Services](#).
4. The mobile agent observing the CWS becomes the global observer; other mobile observers are local.
5. SOAP handlers forward traces to appropriate mobile observers.
6. Local observers analyze these traces and forward them to the global observer.
7. Whenever misbehavior is detected (by global or local observers), correlation then fault location is initiated by the global observer to find the faulty [Web Service](#).
8. The global observer reports to the WSO.
9. The WSO reports to the manager.

Figure 8. Multi-observer deployment

Each local observer is listening to a UDP port to receive events from SOAP handlers. The global observer is listening to two different UDP ports: one to receive events (request or response) from local observers and another port to receive information on detected misbehaviors by local observers. The SOAP handler sends each event between a client and its Web Service to

the attached local observer. The latter forwards this event to the global observer and checks the validity of this event. If misbehavior is detected, the local observer notifies the global observer. Figure 9 shows the overall configuration of interacting client, [Web Services](#), mobile observers, and communication between these entities.

Figure 9. Multi-observer configuration for [monitoring](#) of CWS

Processing CPU and Memory Utilization

Computer resources (CPU and memory) used by traces' collection entities are somehow insignificant with regards to the minimal standard configuration of actual personal desktops/laptops. Except for the mobile agent approach, CPU and memory utilization are so low that they are even difficult to precisely evaluate.

For mobile observers, CPU and memory utilization on a laptop equipped with an AMD Athlon 64/3000+ processor and 512MB RAM, CPU and memory utilization are as follows:

- **Hosting a mobile platform:** if the mobile agent administration interface is located on the laptop, the CPU utilization varies between 2% and 4%. For memory, it uses around 30 Megabytes.
- **Joining a mobile platformⁱⁱ:** if the mobile agent platform is running on a remote computer, joining it requires 12 MBytes memory at the laptop and around 2 MBytes on the host running the administration interface. For CPU, there is almost no impact at both sides.
- **Receiving a mobile observer:** when a mobile observer is received, it requires around 27 MBytes of memory. For CPU, there is a high utilization during 1 to 2 seconds while initializing and displaying the graphical interface of the mobile observer, then the CPU utilization goes back to previous level
- **Processing traces:** even in the worst case where traces are received with a very small delay, the CPU used by the mobile observer for analyzing them is around 2%. However, there is no additional memory utilization.

Network load

The network load introduced by the observation is classified into load due to the deployment of mobile agents and load due to the traces' collection process.

Deployment Load

Since all observers are located at the composite Web Service provider's side, only one mobile agent is generated by the WSO. The size of the traffic to move a mobile agent is around 600 Kilobytes (600 KB).

Trace collection Load

Generally, for each interaction between a Web Service and its client, 2 UDP datagrams are generated: a first datagram from the SOAP handler to the local observer, and a second datagram from this local observer to the global observer. Whenever a local observer detects misbehavior, a third datagram is sent (fault notification). The average size of a datagram is 150 bytes. So, each response/request pair introduces 4 datagrams if everything goes fine, 5 datagrams if one of the events is faulty, or 6 datagrams if both are faulty. We suppose that faults will not occur very often, and then few fault notifications will be generated. This assumption is realistic since all [Web Services](#) are supposed to undergo acceptable testing process before their deployment. The traces' collection load then is reduced to the forward of events, that is, 4 datagrams for a request/response pair. This represents a load of 600 bytes.

Results and Analysis

To illustrate the detection capabilities of our architecture, we injected faults to some [Web Services](#) and/or in the network and monitored the behavior of observers (Table 1). The observers have been able to detect most of the injected faults.

Table 1. Some of the executed scenarios

Target Web Service	Fault description	Comments
CWS	Submit a printDocument request before creating a conference	Fault detected by local and global observer
Call Control	Add a user before creating a conference	Fault detected by local and global observer
Presence	Try to add a user to the conference that is not recognized by the Presence service	Fault detected by local and global observer
Shipping	Request shipping of a document that has not been submitted for printing	Fault detected by local and global observer
Shipping	A trace collection event (shipDocument response) from a handler to the local observer is lost (Figure 10)	Neither the local observer nor the global observer will detect the fault.
Shipping	A trace collection event (shipDocument response) or a fault notification from a local observer to the global observer is lost (Figure 11)	The global observer will not be able to detect the fault or process the notification (correlation)

A fault that cannot be detected occurs when the last event in a communication between a Web Service and its client is lost. As discussed earlier, traces are sent as UDP packets. To be able to detect lost packets and recover the observation, a sequence number attribute is used. An observer detects a lost packet if the sequence number of the following packet is different than expected. When a lost packet carries the last event in a communication, observers will not be able to detect this incident since no future packets will arrive. Table 1 shows brief descriptions of some of the executed scenarios and the reactions of observers (both local and global) to the fault.

Figure 10. trace lost before getting to local observer

Figure 11. trace or fault notification lost before getting to global observer

CONCLUSION

Web Services are new generation of web applications. This new paradigm of communication puts more emphasize on business-to-business interactions rather than the business-to-consumer transactions model that the Internet was largely providing. **Management** of **Web Services** is of prime importance for all entities involved in **Service Oriented Architecture**. In an environment where the interacting components are not known a priori, can be on different operating systems and platforms, coded in different programming languages, the **management** of **Web Services** is very challenging comparing to the management of traditional distributed systems.

In this chapter, we have developed a framework to provide management features to **Web Services** providers and clients by supporting management activities all along the lifecycle of a Web Service, from development to invocation. The framework encourages providers to consider management activities while developing their **Web Services** by specifying **QoWS** attributes. It allows clients to select appropriate **Web Services** using **QoWS** information published previously by providers. Clients make also use of the framework to check if the **Web Services** they are

actually using or planning to use are behaving correctly in terms of **functional** and **non-functional** facets. A prototype of the framework has been developed and used in management of a set of **Web Services**. In this chapter, a conferencing Web Service has been used to evaluate the effectiveness and the overhead of the framework.

As a future work, we intend to enhance the framework by providing support to other management operations such as fault isolation and repair. Work is in progress to implement the remaining components of the framework and to evaluate the overall features of the framework on a large scale **Web Services** environment.

REFERENCES

- 3GPP. (2003). Open Service Architecture WorkGroup. 2003, from <http://www.3gpp.org/TB/CN/CN5/CN5.htm>
- Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., et al. (2003). BPEL4WS Version 1.1 specification. 2007, from <ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf>
- Ankolekar, A., Burstein, M., Hobbs, J. R., Lassila, O., Martin, D., McDermott, D., et al. (2002). DAML-S: Web Service description for the Semantic Web. In S. Verlag (Ed.), *First International Web Conference* (pp. 348-363). Sardinia, Italy: Springer Verlag.
- Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., et al. (2002). Web Service Choreography Interface (WSCl). 2007, from <http://www.w3.org/TR/wsci/>
- Banerji, A., Bartolini, C., Beringer, D., Chopella, V., Govindarajan, K., Karp, A., et al. (2002). WSCl: The Web Services conversation language. 2007, from <http://www.w3.org/TR/wscl10/>
- BEA. (2004). WebLogic platform. <http://www.bea.com>, 2004, from <http://www.bea.com>
- Benharref, A., Dssouli, R., Glitho, R., & Serhani, M. A. (2006). Towards the testing of composed web services in 3rd generation networks. In *IFIP International Conference on Testing of Communicating Systems (TestCom)* (Vol. 3964, pp. 118-133). New York, NY, United States: Springer Verlag.
- Benharref, A., Glitho, R., & Dssouli, R. (2005). Mobile agents for testing Web services in Next Generation Networks. In *MATA 2005* (Vol. 3744, pp. 182-191). Montreal, Canada: Springer Verlag.
- Casati, F., Shan, E., Dayal, U., & Shan, M.-C. (2003). Business - Oriented management of Web Services. *Communications of the ACM*, 46(10), 55-60.
- Case, J., Fedor, M., Schoffstall, M., & Davin, J. (1990). RFC 1157 - Simple Network Management Protocol (SNMP).
- ChangSup, K., Sungwon, K., In-Young, K., Jongmoon, B., & Young-Il, C. (2006). Generating test cases for Web services using extended finite state machine. In *IFIP International Conference on Testing of Communicating Systems (TestCom)* (pp. 103-117). New York, NY, USA: Springer Verlag.
- CPXe. (2005). I3A Standards - Initiatives - CPXe. 2003, from http://www.i3a.org/i_cpxe.html

- Dssouli, R., Saleh, K., Aboulhamid, E., En-Nouaary, A., & Bourhfir, C. (1999). Test development for communication protocols: towards automation. *Computer Networks*, 31(17), 1835-1872.
- Farrell, J. A., & Kreger, H. (2002). Web services management approaches. *IBM Systems Journal*, 41(2), 212-227.
- Fenton, N. E., & Pfleeger, S. L. (1997). *Software metrics : a rigorous and practical approach* (2nd ed.). Boston: PWS Pub.
- Gray, A. R., & MacDonell, S. G. (1997). Comparison of techniques for developing predictive models of software metrics. *Information and Software Technology*, 39(6), 425-437.
- Ho, S., Loucks, W. M., & Singh, A. (1998). Monitoring the performance of a Web service. In (Vol. 1, pp. 109-112). Waterloo, Ontario, Canada: IEEE press.
- HP. (2007). Open View. <http://www.managementsoftware.hp.com>, 2007, from <http://www.managementsoftware.hp.com>
- ISO/IEC. (1989). 7498, Information processing systems -- Open Systems Interconnection -- Basic Reference Model.
- ISO/IEC. (1998). 9596, Information technology -- Open Systems Interconnection -- Common management information protocol.
- Jade. (2007). Java Agent DEvelopment Framework. <http://jade.tilab.com>, 2007, from <http://jade.tilab.com>
- Jiang, Y., Xin, G.-M., Shan, J.-H., Zhang, L., Xie, B., & Yang, F.-Q. (2005). A method of automated test data generation for Web services. *Chinese Journal of Computers*, 28(4), 568-577.
- Kalepu, S., Krishnaswamy, S., & Loke, S. W. (2004). Verity: a QoS metric for selecting Web services and providers. In (pp. 131-139). Rome, Italy: IEEE Computer Society.
- Kreger, H. (2001). Web Services Conceptual Architectures (WSCA 1.0), *White Paper*: IBM Software Group.
- Lee, D., Netravali, A. N., Sabnani, K. K., Sugla, B., & John, A. (1997). Passive testing and applications to network management. In *International Conference on Network Protocols* (pp. 113-122). Atlanta, GA, USA: IEEE Computer Society.
- Leymann, F. (2001). Web service flow language (WSFL) 1.0. 2007, from <http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>
- OASIS. (2005). Universal Description, Discovery, and Integration. 2007, from <http://www.uddi.org/>
- Parasoft. (2006). SOATest. 2006, from <http://www.parasoft.com/jsp/products/home.jsp?product=SOAP>
- Perkins, D., & McGinnis, E. (1997). *Understanding SNMP MIBs*. Upper Saddle River, N.J.: Prentice Hall PTR.
- Ran, S. (2003). A Framework for Discovering Web Services with Desired Quality of Services Attributes. In (pp. 208-213). Las Vegas, Nevada, United States: CSREA Press.

- Salamon, W. J., & Wallace, D. R. (1994). *Quality Characteristics and Metrics for reusable Software*: National Institute of Standards and Technology, .
- Schmietendorf, A., Dumke, R., & Reitz, D. (2004). SLA management - Challenges in the context of Web-service-based infrastructures. In (pp. 606-613). San Diego, CA, United States: IEEE Computer Society.
- Serhani, M. A., Dssouli, R., Hafid, A., & Sahraoui, H. (2005). A QoS broker based architecture for efficient web services selection. In (Vol. 2005, pp. 113-120). Orlando, FL, United States: IEEE Computer Society.
- ShaikhAli, A., Rana, O. F., Al-Ali, R., & Walker, D. W. (2003). UDDIe: an extended registry for Web services. In (pp. 85-89). Orlando, FL, USA: IEEE Computer Society.
- Tsai, W. T., Paul, R., Cao, Z., Yu, L., & Saimi, A. (2003). Verification of Web services using an enhanced UDDI server. In (pp. 131-138). Guadalajara, Mexico: IEEE.
- Vijayananda, K., & Raja, P. (1994). *Models of Communication Protocols for Fault Diagnosis*: Swiss Federal Institute of Technology.
- W3. (2004). Web Services Endpoint Management Architecture Requirements. from <http://dev.w3.org/cvsweb/2002/ws/arch/management/ws-arch-management-requirements.html?rev=1.7>
- W3C. (1999). HTTP. 2007, from <http://www.w3.org/Protocols/>
- W3C. (2001). Web Services Description Language. 2007, from <http://www.w3c.org/TR/wsdl>
- W3C. (2002). *Web Services Management Concern* (White Paper): W3C Consortium.
- W3C. (2003, November). QoS for Web Services: Requirements and Possible Approaches.
- W3C. (2004). Simple Object Access Protocol. 2004, from <http://www.w3c.org/TR/soap>
- W3C. (2006). eXtensible Markup Language. 2006, from <http://www.w3c.org/XML>
- Wahli, U. (2002). Self-study guide: WebSphere Studio Application Developer and web services. 1st. from <http://www.books24x7.com/marc.asp?isbn=0738424196>
- Warrier, U., Besaw, L., LaBarre, L., & Handspicker, B. (1990). RFC 1189 - The Common Management Information Services and Protocols for the Internet (CMOT and CMIP).
- WSDL. (2001). Web Services Description Language. 2007, from <http://www.w3c.org/TR/wsdl>
- Xiaoying, B., Wenli, D., Wei-Tek, T., & Yinong, C. (2005). WSDL-based automatic test case generation for Web services testing. In (pp. 207-212). Beijing, China: IEEE Computer Society.
- Yuming, J., Chen-Khong, T., & Chi-Chung, K. (2000). Challenges and approaches in providing QoS monitoring. *International Journal of Network Management*, 10(6), 323-334.

ⁱ When a composite Web Service invokes a basic Web Service, it is said to be a client of that Web Service.

-
- ii A node can host a mobile observer if it is running a mobile agent platform administration interface or is joining a remote platform.