QUALITY OF SERVICE FOR INTEROPERABLE TRANSACTIONS IN SERVICE-ORIENTED ARCHITECTURE

by

PHUNG HUU PHU

THESIS
Submitted to School of Computer Engineering and Information Technology
University of Ulsan
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

School of Computer Engineering and Information Technology
UNIVERSITY OF ULSAN
June 2006
Quality of Service for Interoperable Transactions in Service-Oriented Architecture

배재수 이명제

이 논문을 공학석사학위 논문으로 제출함
2006 년 06 월

울산대학교 대학원
컴퓨터정보통신공학부 학과
PHUNG HUU PHU
PHUNG HUU PHU 의 공학석사학위 논문을 인준함

심사위원 이명재
심사위원 김명균
심사위원 오훈

울산대학교 대학원
2006년 06월
QUALITY OF SERVICE FOR INTEROPERABLE TRANSACTIONS IN SERVICE-ORIENTED ARCHITECTURE

PHUNG HUU PHU

School of Computer Engineering and Information Technology

APPROVED:

Myeong-Jae Yi, Chair, Ph.D.

Kyun-Myun Kim, Ph.D.

Hoon Oh, Ph.D.
to my

PARENTS

with love
Acknowledgements

First of all, I would like to express my deep gratitude to my advisor, Professor Myeong-Jae Yi for his supports and guidance for my master study. He first brought me to Korea and has provided me a very good environment for my research and study. I am greatly indebted to him for his supports.

I also would like to thank Professor Myung-Kyun Kim, director of Computer Network Lab, and Professor Hoon Oh, director of Ubiquitous Computing Lab. They not only give me exciting lectures and research experience but also provide me valuable comments and suggestions for this thesis.

Over the past two years, the Korean Ministry of Commerce, Industry and Energy, Ulsan Metropolitan City, University of Ulsan, and the Network-based Automation Research Center (NARC) have partially supported my research. They not only grant funding for my work but also support funding for me to attend and present my research results at international conferences in Vietnam, Russia, France. My research could not be carried out without these fundings.

I also acknowledge my lab members, especially Mr. Dea-Seung Yoo, my Korean senior and Jun-Chol Sin, for their attention and help during last two years.

Over the course of my master research, Vietnamese brothers and friends in University of Ulsan, Mr. Pham Van Vinh, Mr. Truong Quoc Thanh, Mr. Trinh Hoang Hon, and others, have given me much encouragement and spiritual support. Without them, my life in Korea would have been more difficult and less enjoyable. The pleasant time with them in Ulsan has led to many memories that would never be forgotten.

Ulsan - South Korea, July 1st, 2006
Abstract

Web Services and Service-Oriented Architecture (SOA) has been playing an important role as a middleware for software interoperability. These technologies are being deployed increasingly in practice for e-business or Enterprise Application Integration. With the success of such deployments, Quality of Service (QoS) properties are in high demands for SOA-based interoperability. Normally, developers use a Web Services framework to build SOA-based applications since such frameworks supply a simple approach for creating, deploying as well as consuming Web Services. Most Web Services frameworks do not considered the aspects of QoS; therefore, it is very difficult for developers to consider these aspects in applications. In this thesis, we present a framework which supports QoS properties for SOA-based transactions. In this framework, transactions can be executed in reliable, effective, and secure manner. The framework is transparent with developers on both client and server side, and supports QoS including performance, accessibility, reliability, and security. The design of our framework provides an easily integrated and flexibly extended approach to SOA-based applications. The framework is built on top of the popular Axis framework such that it automatically inherits Axis’s heterogeneity regarding communication protocols and back-ends of Web Services. The qualitative evaluation of the framework and case studies are also presented to illustrate how our framework can be used in practice.
Table of Contents

Acknowledgements ........................................ vi
Abstract ................................................... vii
Table of Contents ......................................... viii
List of Figures ............................................. x

Chapter
1 Introduction ................................................ 1
  1.1 Problem Statement .................................... 1
  1.2 Goals .................................................. 2
  1.3 Thesis Organization ................................... 4

2 Web Services and Service-Oriented Architecture .......... 5
  2.1 Web Services .......................................... 5
    2.1.1 Basic Web Service standards .................... 7
  2.2 Service-Oriented Architecture ....................... 10
  2.3 Open Issues .......................................... 13
    2.3.1 Web Services security ........................... 13
    2.3.2 Web Services reliability ......................... 16

3 The Proposed Framework Supporting Quality of Service .... 19
  3.1 The Description of the Framework .................... 19
  3.2 The Design of the Framework ......................... 20
    3.2.1 Security module ................................ 21
    3.2.2 Reliability module .............................. 24
    3.2.3 Additional properties ............................ 26
    3.2.4 QoS monitoring .................................. 27
  3.3 The Implementation of the Framework ................ 27
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.1 Apache Axis framework</td>
<td>27</td>
</tr>
<tr>
<td>3.3.2 Apache Axis framework in the context of Remoting Patterns</td>
<td>30</td>
</tr>
<tr>
<td>3.3.3 The implementation of the framework on client side</td>
<td>33</td>
</tr>
<tr>
<td>3.3.4 The implementation of the framework on server side</td>
<td>34</td>
</tr>
<tr>
<td>3.3.5 QoS Monitoring</td>
<td>36</td>
</tr>
<tr>
<td>3.3.6 Performance considerations</td>
<td>37</td>
</tr>
<tr>
<td>4 The Evaluation and Case Studies of the Framework</td>
<td>39</td>
</tr>
<tr>
<td>4.1 The Evaluation of the Framework</td>
<td>39</td>
</tr>
<tr>
<td>4.2 Case Studies</td>
<td>41</td>
</tr>
<tr>
<td>4.2.1 Developing a Web Service on Server Side</td>
<td>41</td>
</tr>
<tr>
<td>4.2.2 Consuming a Web Service on Client Side</td>
<td>45</td>
</tr>
<tr>
<td>4.2.3 A Two-tier Transactions Application Scenario</td>
<td>46</td>
</tr>
<tr>
<td>4.2.4 A Long-running Transactions Application Scenario</td>
<td>48</td>
</tr>
<tr>
<td>5 Concluding Remarks</td>
<td>51</td>
</tr>
<tr>
<td>5.1 Significance of the Result</td>
<td>51</td>
</tr>
<tr>
<td>5.2 Future Work</td>
<td>52</td>
</tr>
<tr>
<td>Publications</td>
<td>53</td>
</tr>
<tr>
<td>References</td>
<td>55</td>
</tr>
<tr>
<td>Curriculum Vitae</td>
<td>60</td>
</tr>
</tbody>
</table>
List of Figures

2.1 The general architectural model for Web Services. . . . . . . . . . . . . . . 6
2.2 Web Services specifications and standards. . . . . . . . . . . . . . . . . . . 8
2.3 SOA-based approach to interoperable transactions. . . . . . . . . . . . . . . 12
2.4 SOAP message security. . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

3.1 The structure of the framework. . . . . . . . . . . . . . . . . . . . . . . . 20
3.2 Sending SOAP message in the proposed security model. . . . . . . . . . . . 23
3.3 The Messaging Model. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
3.4 Architecture of Axis on server side. . . . . . . . . . . . . . . . . . . . . . 28
3.5 Architecture of Axis on client side. . . . . . . . . . . . . . . . . . . . . . 29
3.6 Broker architecture. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
3.7 Basic Remoting patterns. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
3.8 Mapping between Axis framework and remoting patterns. . . . . . . . . . . . 32
3.9 Client objects in Axis. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
3.10 The framework on client side. . . . . . . . . . . . . . . . . . . . . . . . . . 34
3.11 Interaction diagram on client side. . . . . . . . . . . . . . . . . . . . . . . 35
3.12 The framework on server side. . . . . . . . . . . . . . . . . . . . . . . . . . 36
3.13 Interaction diagram on server side. . . . . . . . . . . . . . . . . . . . . . . 37
3.14 QoS Monitoring model. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
3.15 Thread pooling used in the framework. . . . . . . . . . . . . . . . . . . . . 38

4.1 The qualitative evaluation of the proposed framework. . . . . . . . . . . . . . . 40
4.2 Message before dispatching to the framework. . . . . . . . . . . . . . . . . . 41
4.3 Message after dispatching to the framework - part 1. . . . . . . . . . . . . . . 42
4.4 Message after dispatching to the framework - part 2. . . . . . . . . . . . . . . 43
4.5 A two-tier Web Services application using the proposed framework. . . . . 48
4.6 Illustrating a long-running Web Services transaction scenario. . . . . . . . 49
Chapter 1

Introduction

1.1 Problem Statement

Service-Oriented Architecture (SOA) is a new trend in distributed systems aiming at building loosely-coupled systems that are extendible, flexible and fit well with existing legacy systems. SOA is based on Web Services framework, which intends to provide a standards-based realization of the service-oriented computing (SOC) paradigm. SOC is considered as the new trend of distributed computing paradigm that utilizes services as fundamental elements for developing heterogeneous and distributed applications within and across company boundaries. In this new trend, services are offered by service providers, organizations that procure the services implementations, supply their service descriptions and provide related technical and business support [1].

In the SOA-based approach, each system specifies the interoperability with others for the integration and machine-to-machine interactions. The transactions of interoperability are created as services using Web Services/SOAP interface. Services can be published for integration communications thanks to UDDI specification. Each system, depending on particular business rules, might communicate with other systems by request services via SOAP/HTTP transactions.

Although SOA supplies architecture for interoperability in effective manner, there are many challenges that need to be investigated to develop the infrastructure for such transactions. One of the challenges is Quality of Service (QoS) for SOA-based transactions. Normally, developers use a Web Services framework to build SOA-based applications since such frameworks supply a simple approach for creating, deploying as well as consuming
Web Services. However, most Web Service frameworks have not considered the aspects of QoS; therefore, it is very difficult for developers when considering these aspects in their applications. This research deals with the problems of QoS for in SOA-based transactions. This work focuses on answering the question: how to support QoS for interoperable transactions without resorting application layer. Two main aspects of QoS in SOA-based transactions are mainly considered in this thesis: reliability and security. Besides, other properties of QoS such as accessibility, availability and performance in such transactions are also investigated in this work.

1.2 Goals

In our point of view, to support QoS in SOA-based applications, a framework for service management is needed. A common framework identifies specific functions that need to be addressed in order to achieve decentralized interoperability. The framework should be designed in extensible and flexible manner based on open standards supporting Web services transactions. Developers could use the framework by transparently configuring framework mechanisms and could manage services on concrete scenarios.

Recently, there are a number of Web Services frameworks but they were not considered QoS. For instance, Apache Axis framework [16] only provides the way for constructing SOAP processors such as clients, servers, gateways. GLUE framework offers a more simple invocation model by generating an interface automatically and providing it remotely, i.e. only supports client side [14]. Asynchronous Invocation Framework [24], which is built on top of Axis and provides asynchronous invocation on client side without using asynchronous message protocols, or Web Services Invocation Framework(WSIF) [17] support only on client side and did not consider QoS for transactions. Therefore, the goal of this work is to propose a framework that supports QoS properties for SOA-based transactions. The main aim of our intended framework is that the transactions of services are guaranteed the reliability and security. The aspects are transparent with application layer; thus, Web
Services applications built on this framework are guaranteed the reliability and security without considering these aspects in application code. In reliability aspect, the framework deals with the problems system or network failures in transactions of the architecture and resolves the problems that how to manage and recover failed long-running transactions. In the context of security, data transferred inside the framework is guaranteed the integrity, confidentiality, and non-repudiation. Access control of services is also considered via authentication mechanism. Besides, the intended framework also aims to solve the following issues:

- **Provide the better performance of Web Services applications:**
  Performance is influenced by client and server processing times and network latency. Transactions in Web Services-based applications contain a few typical performance bottlenecks compared to other distributed middleware since they requires various message-level processing. Moreover, response time is especially important for conventional, synchronous Web Services; the consuming application is blocked from the time it sends a SOAP request message until it receives the expected response. Therefore, performance property should be considered in the framework.

- **Simple and flexible usage on client side and server side:**
  A simple programming model should be offered to application developers both on client side (service requesters) and server side (service providers). It means that developers only focus on their application layer and should not have to deal with quality of service issues since these issues are supplied by the framework.

- **Support for multiple Web Services implementations:**
  One of the benefits of Web Services is heterogeneity; therefore, multiple implementations for Web Services should be allowed to work with the framework. Moreover, the design of the framework should support the heterogeneity of services, infrastructures, and protocols, as this is the main design goal of Web Services. And as the
technology continues to evolve, a number of specifications are being proposed to address the areas necessary to support Web services. Therefore, the design of framework should provide the extensibility, which is important feature of a framework for extending additional functions in future.

1.3 Thesis Organization

The thesis consists of 5 chapters. The first chapter, which you are reading, is the introduction of the thesis. This chapter states the problems needed to be solve in the thesis, the goals and the structure of the thesis. Chapter 2 introduces the concepts of Web Services and Service-Oriented Architecture. Open issues and challenges of SOA-based applications are also identified in this chapter. In chapter 3, the design and implementation of a framework supporting QoS for SOA-based transactions are presented in details. The evaluation and case studies of the proposed framework are presented in chapter 4 in order to demonstrate the use of the proposed framework in practice. Conclusion remarks of the thesis are present in chapter 5.
In this chapter we introduce the concepts of Web Services and Service-Oriented Architecture, which are technologies that provide a new approach to distributed applications and could overcome the challenges of traditional approach of distributed computing. Open issues and challenges of SOA-based applications are also identified.

2.1 Web Services

Web Service is a new model for using the Web, in which transactions are initiated automatically by a program, not necessarily using browser like the current World-Wide Web. In this new model, services can be described, published, discovered and revoked dynamically in a distributed computing environment. Figure 2.1 shows the general architectural model for Web Services.

In the figure, we can see that the Web Services model consists of three types of participants:

1. **Service providers**: creating Web Services and publishing them to services brokers.

2. **Service brokers**: receiving and maintaining a registry of services published by services providers and providing services information to service requesters for their consuming Web Services.
3. **Service requesters**: searching the registries of service brokers for suitable service providers and then communicating with a service provider to use its services

Web Services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks [1]. eXtensible Markup Language (XML) is a common language used by service requesters and services providers for their connecting and exchanging information in Web Services model. The protocol for communication among services in Web Services model is Simple Object Access Protocol (SOAP). This protocol works over HTTP protocol of the Web. According to [1], we can have a definition of Web Services as follows:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

In the context of interoperability, existing distributed technologies such as CORBA,
COM/DCOM, EJB, Java RMI etc, can be applied but they are strongly coupled the endpoints and therefore could not become pervasive. For instances, Unix RPC requires binary-compatible UNIX implementations at each endpoint, CORBA requires compatible ORBs, RMI requires Java at each endpoint, COM/DCOM requires Windows at each endpoint. Compatibility and standards are the big problems for this heterogeneous interchange system. Moreover, it maybe difficult or impossible to build applications that can communicate via the Internet environment because most these technologies can not work via firewalls.

Thus, the goals of Web Services go beyond those of the classical middleware frameworks such as CORBA, DCOM, or RMI: they aim at standardized support for higher-level interactions such as service and process flow orchestration, enterprise application integration (EAI), and provision of a 'middleware for middleware' [14]. In order to support higher-level interactions, a number of standards are proposed. The following subsections present related Web Service standards and specifications.

### 2.1.1 Basic Web Service standards

This section only introduces briefly basic standards related to Web Services. As presented in previous section, Web Services technology is software components described via WSDL [5] (Web Services Description Language) being capable of being accessed via standard network protocols such as SOAP [7] (Simple Object Access Protocol) over HTTP. Web Services and its related technologies are vital role for communication environment in heterogeneous systems. Web Services are self-contained, modular business process applications that are based on the industry standard technologies of WSDL (to describe), UDDI [8] (to advertise and syndicate), and SOAP (to communicate). These basic standards (i.e., WSDL, SOAP, and UDDI) are all based on XML language [4]. Figure 2.2 shows the stack of Web Services.
XML

Extensible Markup Language (XML) is a markup language in which users can define their own tags for their usage. XML tags convey information about the meaning of data, not just show how data should appear. XML has the advantages in that it can be parsed and processed in platform independently. It can be extended for new applications, supports both unstructured data (documents) and structure data (e.g. from databases). Moreover, XML enables structure sensitive queries and XML-tagged can be mechanically validated.

In general, XML is the key to all the other Web Services standards, because it represents a truly interoperable data representation, allowing disparate applications to communicate across the enterprise or the Internet.
SOAP

Simple Object Access Protocol (SOAP) is a protocol for exchanging information in a decentralized, distributed environment. It defines a mechanism to pass commands and parameters between clients and servers. Similar to Web Services as a whole, SOAP is independent of the platform, object model, and programming language being used. The SOAP protocol is XML-based language consisting of three parts:

- An envelope that defines a framework for describing the message content and how to process it.
- A set of encoding rules for expressing instances of application-defined data types.
- A convention for representing remote procedure calls and responses.

WSDL

Web Services Description Language (WSDL) is a metadata language which is used to define how service providers and service requesters communicate with each other in Web Services applications. WSDL is an XML schema describing a programmatic interface to a Web Service. WSDL service definitions provide documentation for distributed systems and automate the details involved in communications between applications. The definitions include data types, input and output message formats, the operations provided by the service, network addresses, and protocol bindings. Similar to XML, WSDL is extensible to allow for the description of endpoints and their messages, regardless of what message formats or network protocols are used for communication. WSDL provides the flexibility to define software components written in the CORBA, Java/J2EE, or .NET programming models. This layer of abstraction allows SOAP to access to applications written on different operating systems, programming languages, and implementation models. WSDL provides all the information which a client application needs to build a valid SOAP invocation.
UDDI

The Universal Description, Discovery, and Integration (UDDI) [8] is a specification used to describe a mechanism for registering and locating Web Services. As an information database of Web Services, a UDDI server offers a place (registry) to store descriptions about companies and the services they offer in a common XML format. Service requesters can use the UDDI registry to discover and invoke published services. Web-based applications interact with a UDDI registry using SOAP messages. The UDDI is itself a Web Service based on XML and SOAP. Conceptually, the data in a UDDI registry can be divided into three different types of telephone directories:

- A white-pages section, which provides business contact information.
- A yellow-pages section, which categorizes businesses and their services.
- A green-pages section, which provides technical information about the businesses services.

2.2 Service-Oriented Architecture

Service-Oriented Architecture (SOA) is an architectural style whose goal is to achieve loose coupling among interacting software agents. A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Both provider and consumer are roles played by software agents on behalf of their owners [2].

Traditional distributed technologies in which infrastructures and applications were managed and owned by one enterprise are giving way to networks of applications owned and managed by many business partners. The challenges of the interoperability are heterogeneous environment, the desire of fast, effective business process integration, and no disruption to existing back-end system. Since SOA is based on Web Services technologies, which can overcome the weaknesses of distributed technologies; SOA can be applied into heterogeneous system like such transactions. SOA is also a distributed computing environment
and be poised at the inter-section of business and technology, enabling enterprises adapt quickly to change environment [3]. SOA is based on Web Services technology which is based on XML; therefore, it is possible for any platform and programming language to build an application using Web Services. These features of Web Services technology can solve the weaknesses of other distributed technologies such as language and platform dependence, inflexibility, and disruption to existing interfaces of old systems.

Figure 2.3 shows a SOA-based approach to software interoperability. The idea of SOA departs significantly from that of object oriented programming, which strongly suggests that you should bind data and its processing together. The results of a service are usually the change of state for the consumer but can also be a change of state for the provider or for both. Using SOA for interoperable transactions by supplying and calling electrical services for interoperation, the systems can reach the following benefits:

- Each system supplies its electronic services by defining and implementing independently Web Services interfaces based on the existing back-end system; no disruption to the existing software system.

- The systems can communicate with each other by requesting services from the related systems without worrying about platform standards or programming languages. The systems can also adapt functions and services to fit different business processes in an agile, extensibility, flexibility and scalability manner [2]. As business requirements change, it is relatively easy for the IT environment to adapt quickly, which is not the case with traditional legacy systems.

- The integration system can share data, information and knowledge more readily through open standards and common protocols. SOA supports more effective communication both within an enterprise and between an organization and its supply chain since communications that are not hobbled by incompatible systems. This helps create a distinct competitive advantage for all parties involved.
SOA supports security-enhanced environment and identity management; therefore it can guarantee trust and security for interoperable transactions.

In the SOA-based approach, each system specifies the interoperability with others for the integration and machine-to-machine interactions. The transactions of interoperability are created as services using Web Services/SOAP interface. Services can be published for integration communications thanks to UDDI specification. Each system, depending on particular business rules, might communicate with other systems by request services via SOAP/HTTP transactions.
2.3 Open Issues

Although Service-Oriented Architecture provides a promising solution for interoperable transactions, there are various issues that need to be investigated such as security, reliability, transactions work-flow management and so on. In this work, we consider in detail the aspects of security and reliability. This section presents these issues and their prospective specifications and standards.

2.3.1 Web Services security

Web Services involve threats to the host system, the application and the entire network infrastructure. To secure Web Services, a range of XML-based security mechanisms are needed to solve problems related to authentication, role-based access control, distributed security policy enforcement, message layer security that accommodate the presence of intermediaries.

The implementations of Web Services may require point-to-point and/or end-to-end security mechanisms, depending upon the degree of threat or risk. Traditional, connection-oriented, and point-to-point security mechanisms may not meet the end-to-end security requirements of Web Services. However, security is a balance of assessed risk and cost of countermeasures. Depending on implementers risk tolerance, point-to-point transport level security can provide enough security countermeasures.

Traditional network level security mechanisms such as Transport Layer Security (SSL/TLS), Virtual Private Networks (VPNs), IPSec (Internet Protocol Security) and Secure Multipurpose Internet Mail Exchange (S/MIME) are point-to-point technologies. Although traditional security technologies are used in Web Services security, however, they are not sufficient for providing end-to-end security context, as Web services require more granularities. In general, Web Services use a message-based approach that enables complex interactions that can include the routing of messages between and across various trust domains.
Web Services face traditional security challenges is that a message might travel between various intermediaries before it reaches its destination. Recently, there are a number of security standards and specifications for Web Services. Many standards bodies - such as the World Wide Web Consortium (W3C), Organization for the Advancement of Structured Information Standards (OASIS), the Liberty Alliance, and others - are developing horizontal and vertical Web Services infrastructure standards and specifications to allow enterprises to overcome challenges associated with traditional security technologies [1]. One of these standards is Web Services security (WS-Security). The following subsections present briefly such standards and specifications in order to understand and propose an security architecture for Web services in interoperable transactions.

**Message-level security**

In transactions using Web Services, SOAP messages across to multiple boundaries of networks. It requires end-to-end security, from target to destination of the message. Protocol-level only ensure point-to-point transactions; therefore, it cannot ensure secure transactions in Web Services transactions. WS-Security offers the concept of message-level security in which security credentials are embedded into SOAP message. By this way, message-level security can use element-wise encryption so that it can expose some parts of message for routing, hiding critical data from unauthorized parties. Message-level security also persists from originator to processing end-point, for the life of transaction, survives call to external business partner.

**SOAP message security**

This is syntax for concept of message-level security. It defines language-neutral XML formats defined by XML schema. The security mechanisms include use of XML signature to provide SOAP message integrity; use of XML encryption to provide SOAP message confidentiality; attaching and/or referencing security tokens in headers of SOAP messages, carrying security information for potentially multiple, designated actors; and associating
signatures with security tokens—citeurl:XMLSign.

**XML Signature**

This is a standard proposed by W3C and IETF (The Internet Engineering Task Force) in order to supply the XML syntax standard to describe digital signature. With this syntax, document can be sent with digital signature for whole XML document or portions of it as well as other Web resources. The syntax also defines the XML schema for the signature attached to the document. Procedures for computing and verifying signatures and signature survive parsing as well as generation operations are specified in the standard.

XML signature provides integrity thanks to cryptographic integrity check. The integrity can be proven at any time in the life of message; it does not like integrity in HTTPS only for the life of connection. With this standard, we can have integrity of the whole, any subsection, arbitrary data or any combination of these.

**XML Encryption**

This standard offers the encryption procedures and the XML syntax to represent the encrypted data in XML message. With this standard, developers can encrypt an entire XML message, elements, element content, arbitrary data or a combination of these. The detail syntax and related documents can be found at [10].
XML Key Management System (XKMS)

This standard is a proposal from Verisign, WebMethods, and Microsoft with the purpose of creating an architecture using Public Key Infrastructure (PKI) citeurl:PKI. XKMS defines an XML protocol that allows a simple client to obtain key information (value, certificate, management, or trust data) from a Web service. It also describes protocols for distributing and registering public keys, suitable for using in conjunction with the standards for XML Signature and XML Encryption. XKMS helps overcome PKI complexity by allowing Web services to become clients of a key management service [12]. It uses X-KRSS (XML Key Registration Service Specification) to register, revoke or recover key and X-KISS (XML Key Information Service Specification) to locate or verify a key.

Besides, many others standards and specifications have also been proposed and introduced such as Extensible Access Control Markup Language (XACML), Extensible Rights Markup Language (XrML), Web Services Description Language (WSDL), Security Assertion Markup Language (SAML), Liberty Alliance, Digital Signature Standard (DSS), Electronic Business XML (ebXML), WS-Trust, WS-Federation, WS-SecurityPolicy, and WS-SecureConversation.

Web Services technology has been applied into real world and its security challenges must be investigated. Many standards have been issued but they are not the solution of any model or architecture. Depending on the requirements of specific architecture, the combination of WS-Security standards, protocols as well as architectures need to be investigated to secure the specific systems that use Web Services in its transactions.

2.3.2 Web Services reliability

Distributed systems face with failures such as system or network failures. In the context of Web Services, the same problems need to be considered. Dealing with failures is a vital role in the context of a global network linking services belonging to many different organizations. While we cannot eliminate errors in Web Services transactions, the goal
of Web Services is to both reduce the error frequency for interactions and, where errors occur, to provide a greater amount of information about either successful or unsuccessful attempts at service.

The issues of reliability in context of Web Services can be addressed at several distinct levels: the reliable and predictable delivery of infrastructure services, such as message transport and service discovery, of reliable and predictable interactions between services, and of the reliable and predictable behavior of individual requester and provider agents [1].

**Message reliability**

Reliability at the level of messages is often referred to as reliable messaging. In any distributed system there are fundamental limits to the reliability of communication between agents on a public network. However, in practice there are techniques that we can use to greatly increase the reliability of messages, and in those cases where communication fails then we can gain some feedback as to what went wrong [1].

Two aspects should be considered in message level reliability are the possibility of determining that a message is received successfully and exactly once by intended receiver. In the case the receiving of a message is fail, the sender should have ability to resend the message. Although Web Services architecture does not give specific support for guaranteeing the reliability of message or reporting in the event of failures, the structure of SOAP message in Web Services transactions offers the way to accomplish this issue. Headers of SOAP message allow inserting more content which can be used for supporting message auditing the message reliability infrastructures can be deployed. Therefore, it is possible to incorporate overall message reliability within the SOAP message structure. Message reliability is most often achieved via an acknowledgment infrastructure, which is a set of rules defining how the parties to a message should communicate with each other concerning the receipt of that message and its validity [1]. Currently, there are two specifications which support Web Services reliability: WS-Reliability and WS-ReliableMessaging. These specifications are acknowledgement infrastructures that leverage the SOAP Extensibility
WS-Reliability

Web Services Reliability (WS-Reliability) is a specification which is proposed by Fujitsu Limited, Hitachi, Ltd., NEC Corporation, Oracle Corporation, Sonic Software Corporation, and Sun Microsystems, Inc. WS-Reliability is a SOAP-based protocol for exchanging SOAP messages with guaranteed delivery, no duplicate s, and guaranteed message ordering. WS-Reliability is defined as SOAP header extensions, and is independent of the underlying protocol. This specification contains a binding to HTTP. This model enables a sender (i.e., a SOAP node with reliable messaging functions for sending) to send a message to a receiver (i.e., a SOAP node with reliable messaging functions for receiving) that can accept an incoming connection. This specification remains some unresolved functions to accommodate a receiver that cannot accept an incoming connection (e.g., because of a firewall).

WS-ReliableMessaging

This specification (WS-ReliableMessaging) describes a protocol that allows messages to be delivered reliably between distributed applications in the presence of software component, system, or network failures. The protocol is described in this specification in a transport-independent manner allowing it to be implemented using different network technologies. To support interoperable Web services, a SOAP binding is defined within this specification. The protocol defined in this specification depends upon other Web services specifications for the identification of service endpoint addresses and policies.
Chapter 3

The Proposed Framework Supporting Quality of Service

In this chapter we present the design and implementation of a framework which supports QoS for SOA-based interoperable transactions. Two main properties of QoS considered in this framework are reliability, and security. Besides, other properties such as accessibility, integrity, and performance are also investigated.

3.1 The Description of the Framework

As described in introduction section, although SOA supplies architecture for interoperability in effective manner, there are many challenges that need to be investigated to develop the infrastructure for such transactions. One of the challenges is Quality of Service (QoS) of Web Services transactions in SOA. Normally, developers use a Web Services framework to build SOA-based applications since such frameworks supply simple approach for creating, deploying as well as consuming Web Services. However, most Web Services frameworks have not considered the aspects of QoS; therefore, it is very difficult for developers when considering these aspects in their applications. In this work, two main aspects of QoS in SOA-based transactions are considered: reliability and security. Besides, the QoS properties of accessibility, integrity, and performance are also investigated.

To support reliability and security for SOA-based transactions, a framework has been proposed. In this framework, transactions of services are guaranteed the reliability and security. These aspects are transparent with application layer; thus, Web Services appli-
cations are built on this framework are guaranteed the reliability and security without considering these aspects in application code. In reliability aspect, the framework deals with the problems system or network failures in transactions of the architecture and resolves the problems that how to manage and recover failed long-running transactions. In the view of security, data transferred within the framework are guaranteed the integrity, confidential, non repudiation. Access control of services is also considered via authentication mechanism.

### 3.2 The Design of the Framework

Figure 3.1 shows the structure of our proposed model for the framework. There are three main blocks in this framework: interoperability control, security control and reliable control. The purpose of interoperability control is to create SOAP message based on service description and business rules. The SOAP message will be transferred to security control module if a secure exchange of message is required; otherwise, the message will be transferred to reliability control module. The following subsections present the design of these
modules in detail.

3.2.1 Security module

A Web application needs to be defended with a number of threats. Such threats depend on the layer of network such as: Denial of service attack, man in the middle attack, Trojan horse, virus via e-mail, buffer overflow attack, improperly configured client browsers, improperly configured Web and e-mail servers, dictionary attack, brute force attack, smurf (denial of service by flooding the network) attack, replay attack, and domain name server (DNS) attack. Each attack belongs to specific layer including the perimeter, network, host, or the application layer. Each layer has separated defending solution. For example, with perimeter attacks, we can build the firewall, VPN, intrusion detection to defend these threats; with network attacks, access control list (ACL), encryption, intrusion detection can be considered as solutions for these. In this work, the framework is built for Web Services transactions which use SOAP message protocol worked over HTTP protocol. Supposing HTTP rely on a secure network, that mean it warrantees the security in the perimeter, network, host layer, this work focuses on application layer security challenges in interoperable transactions in SOA infrastructure. In this case, SOAP messages need to be protected to ensure a high degree of confidential and integrity. The detail security aspects for interoperable transactions include:

- **Authentication, authorization**: ensuring the SOAP message come from a trusted partner and the permissions for accessing particular services

- **Integrity**: guaranteeing the contents of a SOAP message have not been modified during transactions

- **Confidentiality**: Ensuring nobody has eavesdropped or reached the message contents

- **Non-repudiation**: Ensuring the exact partner sent the message but not anyone else. This property is especially important on business transactions.
Since SOAP messages freely cross network boundaries, travel over any number of possible connections, and pass through intermediaries on their way to their final destination, any response to the challenges must adequately enforce an organization's security policies and minimize the risks to partners in transactions.

Many standards for Web Services security have been issued but they are not the solution of any model or architecture. Depending on the requirements of proposed architecture and related XML Web services security technologies presented, a security model that combines of XKMS, XML Encryption and XML Signature has been proposed for transactions in SOA. This model follows WS-Security by using credentials in SOAP message with XML Encryption and XML Signature. In this model, XKMS is used as a protocol for public key distribution center. The XKMS host plays the role as a Certificate Authority (CA) in interoperable transactions to distribute and to check the trust of public key in XML signature and as a key distribution center for public key cryptography using XML Encryption.

Since using XKMS, which is similar to Public Key Infrastructure, each participant in this model must have key pair (public and private key) and register the public key to a XKMS server. Figure 3.2 illustrates the proposed model in sending procedure.

To clarify the proposed security model, a simple transaction that system A would like to execute a secure transaction with its partner system B is assumed for illustration. Both systems have public key registered on XKMS server. At first, the system A creates a SOAP message following the business rules. Then this SOAP message (XML syntax) will be encrypted by using XML Encryption standard with following steps:

1. Using X-KISS to connect to XKMS host to get public key of B (in the case there is no cache information)

2. Encrypting SOAP message with the agency B public key; this encryption can be calculated the whole SOAP message or the parts of the message depending on the demand of transactions. The encryption algorithm is specified in the SOAP message

3. Signing the SOAP message by its own private key
4. Sending the encrypted and signed SOAP message to the system B via Web services invocation.

After receiving the Web services SOAP message, the system B executes the following steps to check integrity and decrypt the message.

1. With the signed SOAP message, verify the signature of the `<SignedInfo>` element by:

   - Getting the content in tag `<KeyInfo>`, connect to the XKMS server to verify the public key of agency A with X-KISS protocol.
   - Recalculating the digest of the `<SignedInfo>` element by using the algorithm specified in `<SignatureMethod>` tag. Using the public verification key to ver-
ity that the value of <SignatureValue> tag is correct for the digest of the <SignedInfo> tag

- For each <Reference> tag, recalculating the digest contained within the <SignedInfo> element and comparing it to the digest value in corresponding <DigestValue> tag. XML signature has been verified after all comparison is successful.

2. Decrypt data

- For each <EncryptedData> tag, analyzing to specify the algorithm, key information and ciphered data
- Using its own private key to decrypt data following the specified algorithm

Thus, data transferred in this model inherit authentication, data integrity and non-repudiation of PKI architecture and XML signature. It also supplies data confidentiality thanks to XML Encryption. XML information of transactions stored in each system is used to check after transactions, warrantee auditing of security policy. This model fulfills the requirement of Web Services interoperable transactions in SOA architecture.

### 3.2.2 Reliability module

Reliability is one of the important QoS properties for Web Services transactions. This module deals with the problems of failed Web Services requests over time, duplicated request, and the ordered delivery of messages. Since Web Services transactions are transferred normally on HTTP, which provides only best-effort delivery, it is necessary to consider reliability in SOAP message transactions.

As presented in open issues section, the structure of SOAP message offers the way to fulfill reliability by describing additional information and inserting it to SOAP message headers. Although WS-Reliability and WS-ReliableMessaging are two specifications which are proposed to guarantee the reliability of SOAP message delivery, they are much sophis-
In this framework, reliability module is designed on the basic of the Messaging Model [6]. Figure 3.3 illustrates the model. Based on this model, additional fields are added to SOAP message headers in our framework. On client side, these fields are embedded into the request message before sending to Web Services provider:

- **MessageID**: used to identify the request message to avoid the duplication request
- **Sequence Number**: used in the case resend an invocation which has been failed before
- **Time-to-live**: time within which the client program should receive and acknowledgment or result from the server provider from invoking a request.

This module also provides asynchronous invocation for client application for better performance and loose coupling, i.e. the client should not depend on the processing times of the Web Services. To provide these properties, this module will be implemented in multi-thread model for communicating reliably with server.

On server side, upon receiving SOAP message from another module, this module stores reliability information in header fields to manage the reliability. After receiving result returned from appropriate service, the module construct a response SOAP message, inserting
reliability information into SOAP headers. The detail of this module will be described in implementation section.

### 3.2.3 Additional properties

In the context of Web Services, in addition to security and reliability which are solved by proposed modules, performance, accessibility, availability and integrity are other properties of Web Services needed to be investigated. **Integrity** property is gained in security module where data transferred is guaranteed the confidential, integrity and non-repudiation.

**Availability** defines whether a remote object is ready to service requests at a specific time. This QoS property is not tacked by Web Services framework directly, but web servers usually allow one to manage and log the server, which allows for a weak control of availability. Thus, this property does not need to be considered in our framework. However, our proposed framework supports monitoring mechanism, therefore, it allows for a weak control of availability by checking times of unavailability and long response in monitoring log.

**Performance** is an important QoS property in Web Services. Performance is influenced by client and server processing times and network latency. This is a big challenge that software architects and developers pay high attention when deploying a Web Services-based application since Web Services frameworks contain a few typical performance bottlenecks compared to other middlewares. These bottlenecks are formed because XML processing requires various levels of XML parsing and validation. In this framework, more processing time is required in modules, therefore, we apply the multi-thread and thread pooling paradigm to reduce overhead in transactions. The detail is presented in implementation subsection.
3.2.4 QoS monitoring

In distributed systems like Web Services-based applications, controlling application-specific QoS properties on server side such as bandwidth, response time, reliability, or priorities is important. Applications might want to react to changes in the QoS currently provided. To be able to react to QoS changes, monitoring of QoS properties is usually necessary. In this framework, a QoS Monitoring is added to server side in order to monitor request and response messages. This QoS Monitoring module runs independently of web server on server side. On start-up, this module registers with the framework to indicate that it should monitor QoS of the framework. Then, whenever receiving, executing or responding an invocation, the framework sends relevant QoS parameters to this program for monitoring.

3.3 The Implementation of the Framework

In this section, we explain the implementation of modules of the proposed framework. As explained in the previous section, the framework consists of both client and server side. To build a framework for Web Services from scratch is a huge work and is not the main aim of the thesis. Our implementation focuses on the QoS as mentioned, therefore, the implementation should be built on top of a general framework. For that purpose, we choose the popular Apache Axis framework for extending additional functions to support QoS properties as designed. In the next subsections, a brief introduction of architecture of Apache Axis is presented. Then, based on the architecture of Axis, the implementation of our framework is presented on client side and server side.

3.3.1 Apache Axis framework

Apache Axis is a distributed computing framework, much like CORBA, DCOM, RMI, for Web Services. It is an implementation of SOAP 1.x protocol and allows Remote Procedure Call style and messaging call style. The framework shields Web Services developers from
the details of dealing with SOAP and WSDL. Developers could use Axis on the server side to write web service and deploy (using tools in the framework) it as a web application on a web server; also, developers could use Axis on the client side to make writing client to consuming a Web Service.

From architecture point of view, Axis consists of several subsystems working together. Put simply, Axis is all about processing messages. When the central Axis processing logic runs, a series of Handlers are each invoked in order. The particular order is determined by two factors - deployment configuration and whether the engine is a client or a server. The object which is passed to each Handler invocation is a MessageContext. A MessageContext is a structure which contains several important parts: 1) a "request" message, 2) a "response" message, and 3) a bag of properties [16].

On server side, Axis itself is a Servlet implementation; therefore, it can run in any
Servlet Container. Axis has a listener (Transport Listener) and this listener will create a MessageContext and invoke the Axis processing framework. Figure 3.4 illustrates the structure of Axis server.

We can see on this figure that, upon receiving a Web Services request, AxisServlet dispatch to AxisEngine and process SOAP message in MessageContext. Based on the content of SOAPMessage, appropriate handlers are invoked to execute appropriate service on the server. The result is encapsulated into a SOAP message and return to client following the reserved path.

On client side, application code, aided by the client programming model of Axis, will generate a MessageContext and invoke the Axis processing framework. The message path
on the client side is similar to that on the server side, except the order of scoping is reversed.

In either case, the Axis framework’s job is simply to pass the resulting MessageContext through the configured set of Handlers, each of which has an opportunity to do whatever it is designed to do with the MessageContext.

### 3.3.2 Apache Axis framework in the context of Remoting Patterns

Since remoting pattern concept provides about the general, recurring architecture of successful distributed object middleware (such as Web Services), as well as more concrete design and implementation strategies, remoting patterns help system architects extend the middleware with additional functionality. In this thesis, Axis framework is chosen to extend QoS for Web Services transactions. Axis framework did not implement specific QoS such as security or reliability. In this case, remoting patterns could help us to find the best hooks for extending the Axis framework.
Remoting Pattern language is a pattern language that offers a systematic way to reuse software models, designs and implementations to extend, integrate, customize, or build distributed object middleware solutions. Proposed by Uwe Zdun et al. [14], remoting pattern language extends the broker architecture [14] to address the full range of how to use, extend, integrate, or even build distributed object middleware systems.

In broker architecture, see figure 3.6, the communication functionalities are separated from the application functionalities of a distributed system. The broker hides and mediates all communication between the objects or components of a system. Broker architecture consists of a client-side requestor to construct and forward invocations and a server-side invoker that is responsible for invoking the operations of the target remote object. The marshaller on each side handles the transformation of requests and replies that can be sent
over the transmission medium.

In remoting pattern language, there are basic remoting patterns extended from broker architecture. Figure 3.7 illustrates these remoting patterns and their dependencies within broker architecture.

The Interface Description pattern is a specification for a remote object interface. On client side, developers can use this description to construct a client proxy for a given-object type. The Client Proxy pattern is a place-holder for the remote object on client side. By presenting clients an interface that is the same as the remote object’s, the proxy lets client interact with the remote object as if it were a local object. Requestor receives invocations from Client Proxy, then constructs and forwards to the target remote object. The client request handler and server request handler communicate with each other. The client request handler sends the request from requestor to server. The server request handler receives the request and dispatches to invoker.
The Client Proxy pattern is a placeholder for the remote object on client side. By presenting clients an interface that is the same as the remote object’s, the proxy lets client interact with the remote object as if it were a local object.

Requestor receives invocations from the Client Proxy, then constructs and forwards to the target remote object.

The client request handler and server request handler communicate with each other. The client request handler sends the request from requestor to server. The server request handler receives the request and dispatches to invoker, which invokes the appropriate remote object to execute the request.

In the context of Axis framework, remoting patterns are used in its architecture, though the names of patterns are not similar. Figure 3.8 shows the mapping between Axis framework and remoting patterns.

### 3.3.3 The implementation of the framework on client side

In Axis framework, class Service and Call act as Client Proxy patterns that provide local call within the client process to a remote Web Services object. These classes offer the remote object interface and hide networking detail. Figure 3.9 is the relationship among classes in Axis framework.

In our framework, we add more classes based on patterns to Axis in order to control the quality of service for Web Services transactions. The figure 3.10 shows the relationship...
Class ClientQoSHandler is added into Axis to handle the QoS for the proposed framework. This class instantiates class Security, which handles security module, and class Reliability, which handles reliability module and performance processing for client side of the framework. After building MessageContext for an invocation, the framework dispatches the MessageContext to ClientQoSHandler. The handler invokes security module to encrypt and sign SOAP message for the invocation. The algorithm for encrypting and signing is loaded from a configuration file which is exchanged by partners. The message returned from security model is dispatched to reliability class. This class implements the Runnable interface and associates handler object to process the result back to the client thread. Figure 3.11 shows the interaction of class in the proposed framework.

3.3.4 The implementation of the framework on server side

On server side, our framework also is implemented on the top of popular Axis framework. In this implementation, MessageContext received from AxisServer is dispatched to
ServerQoSHandler which is responsible for handling aspects of quality of services within the framework. Figure 3.12 shows the relationship of modules in the customized framework on server side.

The functionalities of additional modules on server side are similar to those on client side. Security module handles MessageContext dispatched from ServerQoSHandler to de-encapsulate (validates the signature and decrypts SOAP message according to designed security model). Reliability module interacts with Handlers on server side. This module implements Runnable interface and be responsible for performance considerations as showed in the following section.

Besides, the framework on server side implements a QoSMonitor module to monitor Web Services execution and performance. Figure 3.13 shows the interactions of object within the proposed framework.
3.3.5 QoS Monitoring

QoS Monitoring module implements QOS OBSERVER pattern in remoting pattern language [14]. This is a daemon program implemented in TCP socket listening events from the framework on server side. This program can run on the same host with the framework (web server) or another host to reduce computation power. On startup, this program registers to the framework by writing parameters on a configuration file. The file is loaded by the framework to determine which events, such as invocation start/invocation end, connection establishment, incoming invocations, return of invocation, marshaling start/end, should be reported to the QoS monitoring module. Figure 3.14 shows the implementation of QoS Monitoring module for the framework.
3.3.6 Performance considerations

As presented before, to reduce the overhead in the framework, multi-thread and thread pooling mechanism is used in implementation both on client and server side.

On client side, multi-thread technique provides an asynchronous invocation so that the framework can improve performance thanks to the client could resume its work after sending an invocation.

On server side, on receiving a request, the request message is dispatched to the QoS Handler before invoking the appropriate remote object; a new thread is instantiated to process QoS properties as designed. The multi-thread mechanism can be used in order to process concurrent invocations, however, it also incur more overhead due to instantiating threads. The thread pooling technique [15] can be used to reduce this overhead. In this
technique, threads are shared in a pool, when needing the handlers can get a thread-worker from the pool and then release a thread back into the pool when finishing. The pool eagerly acquire a pre-defined number of workers. If the demand exceeds the available resources, the pool would instantiate more resources to execute. Figure 3.15 shows thread pooling model used in the framework.

Figure 3.14: QoS Monitoring model.

Figure 3.15: Thread pooling used in the framework.
In this chapter we give the qualitative analysis and evaluation of the framework. We also present how the framework can be used in practice. Concrete scenarios are built to demonstrate the use of the framework. Simulation results and properties are also presented and discuss in this chapter.

4.1 The Evaluation of the Framework

The figure 4.1 gives qualitative evaluation of properties of the proposed framework as well as the comparison with other related frameworks. As designed and implemented, the framework is based on Axis and provides the properties of QoS that Axis did not support such as security, reliability, performance and QoS monitoring. Compared to others related frameworks showed in the figure, our proposed framework gains better support for Quality of Service properties.

In the context of security, the framework provides the authentication, integrity and non-repudiation for transactions thanks to that data is signed using XML Signature and keys are certified by XKMS system. The confidential of data is also ensured by XML Encryption. In this framework, various cryptography algorithms can be used by editing configuration files, without rebuilding the framework.

In the context of reliability, the framework provides fault-tolerant for transactions in SOA-based applications. The framework on client side could monitor requests, and guar-
Figure 4.1: The qualitative evaluation of the proposed framework.

Figure 4.2 is a raw SOAP request message without considering about QoS properties. Figure 4.3 and Figure 4.4 show the SOAP message processed by the framework before transferring the message over network.

From the implementation view, the framework is built on top of Axis framework such that we can use all functions of Axis. Axis framework is implemented in Java language; therefore, it can run in any application server supporting Java. In demonstrations, we install our framework (customized Axis) on Apache Tomcat Server 5.5 running Windows XP Professional platform. The core of framework is file axis.jar, packing from Axis source.
Figure 4.2: Message before dispatching to the framework.

files and additional Java classes. Besides, numerous Java libraries (for example, jaxrpc.jar, saaj.jar, commons-logging.jar, commons-discovery.jar, wsdl4j.jar) are needed to run this framework. When these libraries are declared on a server, such as Tomcat, the framework is ready to use.

4.2 Case Studies

4.2.1 Developing a Web Service on Server Side

On server side, developers could construct a Web Service by writing a program in Java. Let look a simple example:

```java
import java.util.*;
import java.text.SimpleDateFormat;

public class DateService {
    public String getDate(String format){
        Date today = new Date();
        SimpleDateFormat formatter = new SimpleDateFormat(format);
        String result = formatter.format(today);
        return result;
    }
}
```
Figure 4.3: Message after dispatching to the framework - part 1.
Figure 4.4: Message after dispatching to the framework - part 2.
public String getDate()
{
    return getDate("d MMM yyyy");
}

After compiling this from into a class file, developers could deploy this program to the framework as a Web Service. Axis provides a Web Service Deployment Descriptor (WSDD) file for deployment a Web Service. The below code is a simple deployment descriptor for a basic service:

```xml
<deployment xmlns=http://xml.apache.org/axis/wsdd/
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
    <service name="DateService" provider="java:RPC">
        <parameter name="className" value="simpleService.DateService"/>
        <parameter name="allowedMethods" value="*"/>
    </service>
</deployment>
```

To deploy this service to the framework, we can use the AdminClient tool in Axis by executing the tool in Java Runtime Environment:

```bash
% java org.apache.axis.client.AdminClient -p 8080 deploy.wsdd
<Admin>Done processing</Admin>
```

When deploying successfully, the service is ready to be consumed on server side. Here, we can invoke the deployed Web Services via URL:

`http://localhost:8080/axis/services/DateService`
As described in chapter 3, once a URL is accessed by SOAP client, the INVOKER in the framework is responsible for handling and invoking a remote object (for example, for above URL, class simpleService.DateService is invoked by the framework). In the view of developers, they do not need to pay attention in how the service will be invoked since the mechanism is provided by the framework. In our customized framework, the QoS properties are guaranteed in the framework for Web Service transactions without the investigating of developers.

4.2.2 Consuming a Web Service on Client Side

To consume a Web Service on client, developers have to write programs that construct a request to the service. Of course, a Web Service client can be written in any programming language implemented SOAP protocol. To do this, developers must process many difficult works such as getting the WSDL file, construct a SOAP message, connect to server and send the SOAP message, etc... The Axis framework enables developers to escape these steps by providing a REQUESTOR that developers can use to construct requests dynamically. The details of transporting the invocation across the network are handled by the Axis framework. Our framework is based on Axis, therefore, users could use our framework in the similar way, without worrying about lower layers and QoS since they are provided by our framework.

The below simple program illustrate a client program uses the REQUESTOR to invoke a remote Web Service:

```java
public class DateClient {
    public static void main(String[] args) throws Exception{
        String formatString = args[0];
        Object[] arguments = null;

        String endpoint =
```
"http://localhost:8080/axis/services/DateService";
Service service = new Service();
Call call = (Call) service.createCall();
call.setTargetEndpointAddress( new java.net.URL(endpoint));
call.setOperationName( "getDate");
if(formatString!=null) {
    call.addParameterter("format",XMLType.XSD_STRING,
        ParameterMode.IN);
        arguments = new Object[]{formatString};
}
call.setReturnType(XMLType.XSD_STRING);
String result = (String)call.valueOf(arguments);
System.out.println("getDate service from a WS: " + result);
}
}

Using the Axis framework, developers on both client and server side only need to investi-
gate on their application layer without worrying the details of invocation in lower layers.
On client side, the client code could consume a Web Service like calling a local object; on
server side, developers only need to write an application and deploy it to Axis framework.
Our framework is customized from Axis, therefore, users could use our framework in the
same way as Axis. Of course, QoS properties such as security and reliability of transactions
are guaranteed in our framework.

4.2.3 A Two-tier Transactions Application Scenario

Let consider a scenario in Internet banking transactions. A Bank A provides the online
application for its customers to perform banking transaction via Internet environment. A
customer of bank A would like to transfer an amount of money from her account to her
partner’s account in bank B. In this case, the system of bank A must communicate with that of bank B to perform the transaction. Based on the benefits of Web Services technology, banks develop their services which are used by their partners. Data transferred in such transactions must be ensured the confidentiality, integrity and non-repudiation. Moreover, such applications require high reliable properties.

In practice, banks can build Web Services interfaces by negotiating specifications for transactions such as security, transactionality, reliability, etc... This work needs to spend a lot of time and the implementation is very difficult to extend. In this case, the deployment of common framework between application layer and service might be used. For this purpose, our proposed framework can be deployed among banks so that Web Services banking applications can be built on top of our framework. Since the framework is designed and implemented in Java, it can be deployed in any web server supporting Java. Thus, the framework can be used in multiple platforms since Java language is multiple platform programming language. Once using the framework for Web Services applications, transactions transferred among systems are guaranteed QoS properties provided by the framework. Thanks to extensibility and flexibility of design and implementation of the framework, additional properties are easily added to the framework if the system needs more QoS properties or other aspects.

The use of framework is presented in previous sections. Figure 4.5 illustrates the interactions of the described application.

Back to the scenario, when a user would like to make a account transfer command from bank A site, she enters required information and then submits. Since this information is confidential, data transferred on Internet must be keep the secret. Since data from a web client to web server is transferred outside our framework, the security of data could be guaranteed by using Secure Socket Layer (SSL) over HTTP. When the web application receives data from the user, it constructs the remote service invocation by using our framework. Data is automatically constructed into a SOAP message by the framework, and then modules add more fields into the message to control the QoS. The SOAP message
is dispatched to transport handlers and transfer to server. The server on receiving a message dispatches the message to the framework for QoS controlling and invoking the service. Result of invocation is sent back in reserved path. These procedures are transparent with developers. Since using the proposed framework, data transferred within our framework is guaranteed security, transactions are guaranteed reliability.

4.2.4 A Long-running Transactions Application Scenario

In e-business systems, transactions are usually transferred in multiple machines boundary. For example, in an Internet-based credit card payment system, credit card information is transferred via at least three sites. When a customer submits her credit card information for payment via a web site, the data is transferred to web server, and then the web application connects to a payment gateway to perform the credit card charge. The payment gateway does not process this information but dispatches this data to an appropriate system of card issuer. On receiving the data, the card issuer’s system might process data or dispatches data to its partner depending on the card policy. The result of the transaction is sent
back in reserved path to the customer. We can figure out that this is a long-running transaction in practice. In these long-running transactions, response time and reliability of transactions are important aspects. Moreover, transferred data passes through different organizations’ system, therefore, security should be considered in message-level. From these requirements, our framework could be applied into such transactions in practice. In this scenario, the system of e-commerce site, payment gateway and card issuer could install our framework to use for such transactions. Since security algorithms are set in a configuration file, organizations can negotiate security policy and set the configuration file without customizing the framework. The framework also provides the reliability and other properties of QoS, such as availability, accessibility and performance for Web Services transactions. The e-commerce site uses the client side of framework, constructs a service invocation to request to the payment gateway system. The payment gateway system uses the server side of the framework to provide payment service for e-commerce site systems. Then it uses the client side of framework to perform the request to the system of card issuer. The framework is used in card issuer system on server side to provide credit card charging for payment gateway systems. Figure 4.6 shows the transactions of this scenario.

![Diagram](image-url)

Figure 4.6: Illustrating a long-running Web Services transaction scenario.

In this scenario, security is guaranteed in message-level and can be configured in a file without customizing the framework. The properties of security such as integrity, confi-
dentiality and non-repudiation are guaranteed in transitive manner through point-to-point transactions between each two contiguous systems. Reliability of transactions is also guaranteed in point-to-point manner in a long-running transaction. The framework provides asynchronous invocation on client side; therefore the performance penalty of such a long-running transaction could be reduced. Clients can significantly resume it work faster when transaction is dispatched to a next system. This aspect is especially important in the case of reactive server applications such as the payment system in this scenario. Multi-thread and thread pooling mechanism on server side of the framework offers the better performance for Web Service transaction processing.
Chapter 5

Concluding Remarks

5.1 Significance of the Result

Provision of Quality of Service (QoS) is open challenge when deploying SOA-based applications. In this thesis, a novel Web Services and Service-Oriented Architecture and their open issues in the aspects of QoS have been studied. A Web Services framework has been proposed to support the properties of QoS for Web Services transaction in SOA infrastructure.

Two main aspects are considered in the framework: security and reliability. Besides, the framework also supports other properties of QoS for SOA-based applications such as availability and performance. Since the QoS aspects are transparent with application layer, the supported properties of QoS are guaranteed in Web Services applications built on this framework without explicit consideration in application code.

The framework was designed with remoting pattern language for distributed object frameworks. In this way, the design of framework allows flexible extension with additional properties and Web Services specifications. The framework is implemented on top of the popular Axis framework; thus, the framework automatically inherits Axis’s heterogeneity regarding communication protocols and back-ends of Web Services.

In addition, the framework on client side is designed and implemented in multi-thread mechanism so that a client can significantly resume with its work faster. Thus, the performance penalty of Web Service can be reduced, especially in long-running transactions of SOA-based applications. On server side, the QoS properties of availability and performance are also improved by using thread pooling technique. Scenarios based on top of
our framework have been developed. These scenarios demonstrate that the framework can work well in practice and it supports the properties of QoS for SOA-based applications.

5.2 Future Work

In this thesis, we have proposed and developed an approach for providing Quality of Service, which is in high demands, for SOA-based software interoperability. However, providing end-to-end QoS in a heterogeneous, potentially cross-organizational SOA is a particular challenge and the limits and benefits have still to be investigated. In the future, we will study on Quality-of-Service for SOA-based applications both on qualitative and quantitative. We will refine and evaluate performance and QoS properties in particular application areas so that these aspects can be applied in real world SOA-based applications. QoS negotiation and agreement, contracts, composition of QoS requirements will be also considered in future work.
Publications

Publications related to this work


Other Publications


Papers under reviewing

References


http://www.w3.org/TR/wSDL

http://docs.oasis-open.org/wsrM/ws-reliability/v1.1/

http://www.w3.org/TR/soap12-part1/.

[8] OASIS Open, ”Universal Description, Discovery, and Integration (UDDI)”, 2006
http://www.uddi.org/


[32] Mark Colan, ”SOA and Web Services: Where we are, where we’re


[34] P. Baglietto et al., ”Deployment of Service Oriented Architecture for a business community,” Proceedings of the Sixth International Enterprise Distributed Object Computing Conference (EDOC’02), September 17 - 20, 2002. Lausanne, Switzerland, pp. 293–304.


Curriculum Vitae

Phung Huu Phu was born in May 25, 1979, in Thua Thien Hue province, Vietnam. He graduated from Quoc Hoc High school, Hue City in 1996. He entered Ho Chi Minh City University of Technology in September 1996 and awarded the bachelor degree of Computer Engineering in March, 2001.

After graduation, he has worked as an assistant lecturer at the Faculty of Information Technology, Ho Chi Minh City University of Technology since April, 2001. He was promoted lecturer in June, 2003.

In the fall of 2004, he got a scholarship and entered the Graduate School of Computer Engineering and Information Technology, University of Ulsan, South Korea. While pursuing a master’s degree in Computer Science he worked as a Research Assistant at Applied Software Engineering (ASE) Lab, under the supervision of Professor Myeong Jea Yi. Phu was a member of the Korea Information Processing Society (KIPS).

Permanent address: Thuy Duong, Huong Thuy District,
Thua Thien Hue Province, Vietnam
Email: phunghuuphu@gmail.com

This thesis was defended in public in May 25, 2006 at the School of Computer Engineering and Information Technology, University of Ulsan.