MONASH University
NETWORK Computing

"Adding Session and Transaction Management to XML Web Services by Using SIP"

Submitted in partial fulfillment of the requirements for the degree of

Master of Network Computing (Minor Thesis)

By

Wei Dong

(ID:19082983)

Supervisor: Associate Professor Jan Newmarch

Submitted February, 2005

Declaration

I, Wei Dong (**19082983**), being a student at Monash University, do declare that the minor thesis being submitted by me for an assessment:

- is my own work and has not been copied from any other sources,
- has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education,
- affirm all information derived from the published and unpublished work of others has been duly acknowledged.

Wei Dong
February 28th, 2005

Acknowledgements

Firstly, I thank my supervisor Associate Professor Jan Newmarch for his patiently guidance and support throughout the project. Also, his kindness help on both writing the thesis and my studying life have brought me great encouragement to accomplish this thesis.

Secondly, I would like to thank Judith Rochecouste and Bennos Cousins for their assistance during writing my thesis.

Thirdly, I would like to thank my husband Xiaobo Liang, my parents and parents-in-law. Without their love and supports, I would not have pursued my dreams.

Finally, this thesis is dedicated to my incoming daughter. The exciting feeling she brings to me could not be expressed in words. I wish that she will own a healthy and happy life.

Abstract

In recent years, Web services have been drawing more attention to the computer industry. They are built on open standards and provide solutions for integrating applications across enterprises. The technologies behind Web services are Universal Description, Discovery and Integration (UDDI), Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP).

SOAP, which is responsible for delivering messages between applications, is a connectionless protocol. In other words, SOAP cannot keep the session state between SOAP calls. However, most E-commerce applications need to maintain the session state and transaction information between Web services.

In this thesis, I propose and develop a new mechanism that supports Web services session management by using Session Initiation Protocol (SIP) which is an IETF standard session control protocol. In addition, based on this session management mechanism, a simple transaction management solution for Web services is proposed. Two systems are implemented as demonstrations for both mechanisms in this thesis. Moreover, more functions can be added in these systems, such as security, service registry and discovery.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Overview

The Internet has brought great changes to our daily lives in the last few decades. E-mail has taken the place of paper communication. Online conferences allow people to talk just as in face-to-face meetings even they are long distance away. The Internet collects huge amounts of information which enables people to know events which have happened throughout the world without leaving their home. Some people now describe the world as a global village.

People are no longer only satisfied with surfing static HTML (Hypertext Markup Language) Web pages. They prefer interactive activities and Web pages that are generated to handle the information they input. Fortunately, dynamic Web page technologies have made this possible, although some of these interactions just involve inputting and submitting simple HTML forms.

With the development of network and distributed programming technologies, more and more software services are added in the internal network. For instance, airline companies can provide timetable services for aircrew and booking clerks. If these services could be integrated across enterprises and published on the Internet, it would bring considerable benefits to many companies. However, this requirement can still be out of the reach of dynamic Web page technologies, as different companies often use different technologies to implement their Web sites. In addition, it is difficult to accomplish application integration across enterprises without open standards. Recently, Web services which are built on open standards have been garnering much attention in the computer industry. Web services offer convenient standard ways to open up the functionality between different applications and to provide solutions when executing business transactions across enterprises.

The technologies behind Web services are Universal Description, Discovery and Integration (UDDI), Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP). UDDI provides mechanisms for Web services registration and finding; WSDL is responsible for describing Web services, and SOAP can be used to carry remote method calls and their replies. These technologies have been standardized by the World Wide Web Consortium (W3C) and generally accepted by computer industry.

Although Web services are widely used in industry projects, some problems have arisen. One is that SOAP, which is based on an XML (Extensible Markup Language) format to transport messages between applications, is a connectionless protocol. In other words, SOAP cannot keep the session state between SOAP calls. However, in most E-commerce applications, such as a "shopping cart" or transferring money between banks, the session and transaction states need to be maintained. The traditional techniques: cookies, URL rewriting and hidden form information are not suitable to manage SOAP sessions and transactions. Consequently, new mechanisms need to be invented to manage sessions and transactions for Web services.

1.2 Research Questions

The objectives of this thesis are investigating new mechanims to use an IETF (Internet Engineering Task Force) standard session control protocol, Session Initiation Protocol (SIP), to manage sessions and transactions for Web services.

The major research questions addressed by this thesis are:

Question 1 is whether SIP can be used to manage sessions and transactions for Web services.

Question 2 is how to inform Web services the beginning and the ending of a Web service session by using SIP.

Question 3 is how to manage multiple sessions and transactions on the Web service server.

Question 4 is how to keep the session state between the client application and the Web service server during communication.

If SIP can be used to manage sessions for Web services, it will provide a standard manner for establishing, modifying and terminating a Web services session. This thesis proposes a mechanism of using SIP to manage Web services sessions. Based on this mechanism, a simple solution for Web service transaction management is provided. The research hypotheses and the implementation gaps of using SIP to manage sessions and transactions on Web services are discussed in Chapter 3.

1.3 Thesis Structure

The structure of this thesis is as follows:

Chapter 2 describes the background knowledge of Web services and its three components UDDI, WSDL and SOAP. This chapter also reviews the most frequent session management techniques used for Web-based applications and their limitations when applied to manage SOAP sessions. Finally, a brief introduction to SIP and its related protocol Session Description Protocol (SDP) are provided.

Chapter 3 presents the current status of Web services transaction management and then the research hypotheses and the session and transaction management implementation gaps are listed. Following that, this chapter proposes the aims of this thesis. It also discusses two systems, an Online Video Shopping Web Services System and an E-banking Web Services System, which are developed as demonstrations in this thesis.

Chapter 4 investigates the mechanism of using SIP to manage sessions on Web services. In this chapter, the Online Video Shopping Web Services System is developed. The supporting technologies used in the implementation are also briefly described.

Chapter 5 investigates the mechanism of using SIP to manage transactions on Web services. Based on the previous system, the E-banking Web Services System is implemented in this chapter.

Chapter 6 first evaluates the mechanisms proposed in this thesis, and then the research summary is provided. Finally, it outlines possible future work.

1.4 Summary

The three components, UDDI, WSDL and SOAP, enable Web services to integrate applications running on different platforms. However, there is no formal definition for managing sessions on SOAP messages. In most E-commerce applications, the session state needs to be maintained. Thus, new mechanisms need to be invented to manage sessions for Web services. The contributions of this thesis are: firstly, instead of creating a new protocol to manage sessions for Web services, it uses a standard session control protocol — SIP to accomplish this goal which enhances the extensibility of Web services. Secondly, based on the proposed mechanism, this thesis provides a simple solution for Web service transaction management which conforms to the two-commit protocol and can be easily extended by adding more Web services and security solutions. Two demonstration systems — an Online Video Shopping Web Services System and an E-banking Web Services System are implemented to accomplish the proposed session and transaction management mechanisms in this thesis.

Chapter 2 Background Knowledge

2.1 Introduction

With the rapid development of computer network technology, distributed programs have been used for information exchanging and services sharing. These distributed program technologies include Sun Remote Procedure Call (RPC) system, Microsoft Component Object Model (COM), Distributed Component Object Model (DCOM) and Common Object Request Broker Architecture (CORBA).

However, when these programs are applied widely, some of drawbacks are observed. For example, in Sun's RPC system, a server might provide many services. After a client sends a request to the server with specified service name and parameter values, the server would send back a response value from one of its services. The response value which is represented in binary form is dependent on operating system, hardware, and programming languages. Although Sun specified a standard binary format to encode the parameter values, representing the values is still a problem. In addition, the proprietary ownership of Microsoft's COM and DCOM and CORBA's confusing IDL (Interface Definition Language) and complex ORB (Object Request Broker) system have driven programmers to find a standard, cross-platform, language-independent technology to implement distributed system. The advent of Web services makes these dreams into reality [1].

This chapter is structured as follows. Section 2.2 provides the background introduction of Web services and the technologies behind it. This is followed by an overview of frequently used session management techniques in Web-based applications in section 2.3. Section 2.4 introduces Session Initiation Protocol (SIP) and its messages. Section 2.5 provides a brief introduction of Session Description Protocol (SDP). Finally, section 2.6 summarizes this chapter.

2.2 Web Services

In this thesis, the term Web service refers to a software system which supports the interoperable interaction between different machines over a network. Web service makes use of WSDL to describe its interface. Other systems can invoke Web services by using SOAP messages, which is typically transported by HTTP (Hypertext Transfer Protocol) [2].

Web services are based on a Service-Oriented Architecture (SOA) where all the software systems are distributed as a set of services. In order to allow these Web services to be used by other systems, there needs to be a formal mechanism to describe, discover and invoke services. Figure 1 illustrates the three basic roles and the interactions between the roles in the Service-Oriented Architecture [3].

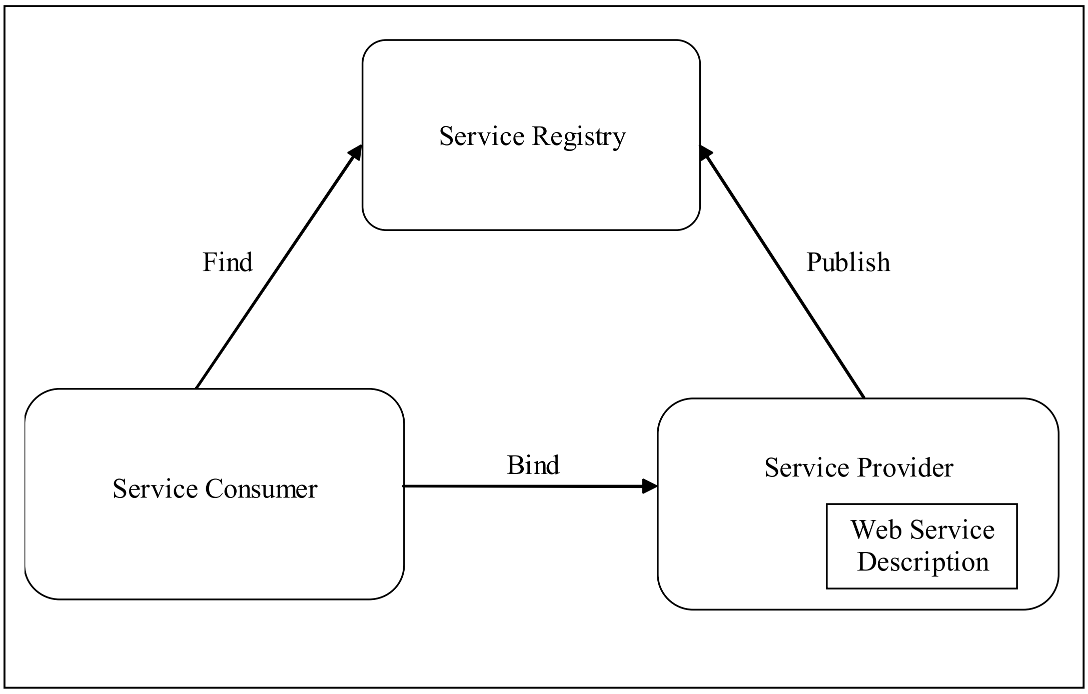


Figure 1 Roles and Their Interactions in a Service-Oriented Architecture

From Figure 1, it can be seen that a Service-Oriented Architecture has Service Provider, Service Registry and Service Consumer three roles. The service provider's main task is to implement the functions of the Web service. After that, the provider should use a standard way to describe the interface for these functions. Finally, the interface should be published to a service registry to allow the service consumer to discover the services [3].

The service registry can be regarded as a Web services library. It accepts all Web services published by service providers, and displays them to service comsumers. Through this way, the service comsumers can find the information on how to bind and invoke the Web services [3].

The service consumer utilizes the Web services offered by service providers to gain useful information. Firstly, the service consumer extracts the service interface information from service registry. From the services interfaces, the service consumer will know the method names, parameter data types and transport protocols to invoke the method. Then it uses this information to bind with the Web services. Finally, the service consumer can invoke the Web services. The technologies supporting these three basic roles are UDDI, WSDL and SOAP.

2.2.1 UDDI

Universal Description, Discovery and Integration (UDDI) acts as the service registry in Web service Service-Oriented Architecture. It specifies how a Web service registers its service information to a service depository and allows service consumers to find those registered services and to use them [4]. In other words, UDDI supplies a register mechanism for distributed Web services and offers a way for service consumers to find the services. Its function is similar to the Naming Services of CORBA and RMI (Remote Method Invocation) [5].

2.2.2 WSDL

Web Services Description Language (WSDL) is an XML based format to describe

Web services as collections of communication end points that can exchange certain messages [6]. When a Web service publishes its service to a UDDI registry, it is the WSDL document which is used to describe that Web service. In fact, the WSDL document provides a Web service's interface for the user to bind and communicate with the service, just as the IDL of CORBA [7].

2.2.3 SOAP

- **Definition and Structure**

Simple Object Access Protocol (SOAP) is a lightweight protocol which allows applications to exchange information in a decentralized, distributed environment. It is XML based format to construct messages, which can be not only exchanged over a variety of protocols but also independent of particular programming languages [8].

The following figure is a pictorial representation of the SOAP message structure. (see Figure 2)

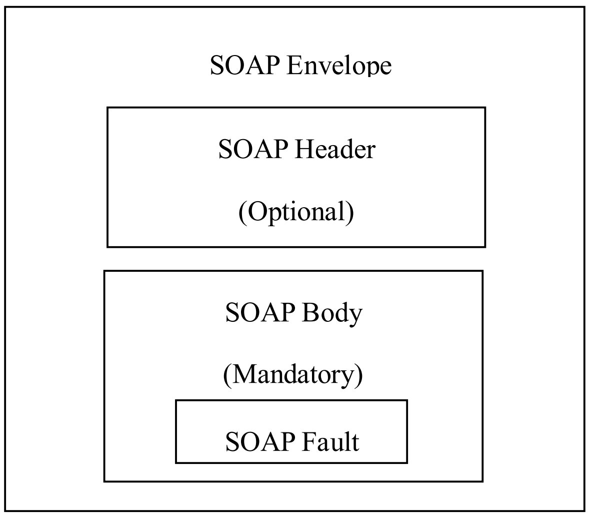


Figure 2 The SOAP Message Structure

From Figure 2, it can be seen that a SOAP message consists of an envelope, an header and a body. The envelope, as the top level element of the XML document presenting in the SOAP message, contains a local name, a namespace declaration, zero or more namespace qualified attribute information items, and one or two child elements — an optional header and a mandatory body [8]. Although the SOAP header is an optional child element in the SOAP envelope, it provides greater extensibility to SOAP messages, thus allowing applications to extend the messages by adding information, such as authentication and transaction management, to the header element [9]. The SOAP body is a mandatory child element in the SOAP envelope. It offers a mechanism for exchanging information with the SOAP message recipient. It may have a child element called "SOAP Fault" which is used to carry error and status information [8]. To summarize, the function of the SOAP body is encapsulating RPC calls and reporting errors.

- **SOAP messages example in Remote Procedure Calls (RPC)**

The W3C SOAP 1.2 Part 0: Primer document [10] claims that using XML to encapsulate remote procedure call functionality is one of the design goals of SOAP Version 1.2. Another part of this document "SOAP Version 1.2 Part 2: Adjuncts" [11] defines a standard for the RPC invocations and responses carried in SOAP messages. Figure 3 and 4 are examples of SOAP messages embedded in an HTTP request (Figure 3) and a corresponding HTTP response (Figure 4). They demonstrate how SOAP makes use of XML vocabulary to encode parameters and to return values in its messages.

```
POST /axis/ProductsPriceQuery.jws HTTP/1.0

Content-Type: text/xml; charset=utf-8

Accept: application/soap+xml, application/dime, multipart/related, text/*

Host: 127.0.0.1

SOAPAction: ""

Content-Length: 437


<?xml version="1.0" encoding="UTF-8"?>

<soapenv:Envelope

      xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/

      xmlns:xsd="http://www.w3.org/2001/XMLSchema"

      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <soapenv:Body>

          <getProductPrice

                    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

                    <productName xsi:type="xsd:string">watermelon</productName>

          </getProductPrice>

  </soapenv:Body>
```

Figure 3 A SOAP Message Embedded in an HTTP Request

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Server: Apache-Coyote/1.1
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
        xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
      <getProductPriceResponse
                soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
            <getProductPriceReturn xsi:type="xsd:string">
                      watermelon price is $2 per kilo
            </getProductPriceReturn>
      </getProductPriceResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 4 A SOAP Message Embedded in an HTTP Response

In this example, a Product Price Query Web service is created to answer the *getProductPrice* SOAP requests. The first six lines in Figure 3 and the first four lines in Figure 4 are the standard HTTP header. The SOAP message, as the HTTP message payload is presented in the body. From the SOAP request message, it can be seen that the SOAP body carries a product name “watermelon” as the parameter to query its price. After the server invokes the request’s remote procedure, the Product Price

Query application gets the price of the watermelon and then inserts the answer in the *getProductPriceResponse* tag of the SOAP response message. Note that the SOAP header element is omitted in this example.

- **The advantages and disadvantages of using SOAP in Web services comparing to other distributed computing techniques**

SOAP possesses many advantages Compared to existing distributed computing techniques, such as Sun's ONC RPC (Open Network Computing Remote Procedure Call)[12], Microsoft's DCOM, CORBA's Internet Interoperable ORB Protocol (IIOP) and Java RMI.

Firstly, SOAP enables universal communication between applications and services through the Internet. Anyone who has tried to integrate programs across DCOM and CORBA/IIOP should know that doing so is time consuming and expensive [13]. Unlike CORBA/IIOP, DCOM and RMI protocols, which use binary formats for the remote service invoking, SOAP, with the help of XML, uses text format (Unicode) to organize the exchanging data [14]. Because XML is a text-based, platform and language independent method of describing data, it enables SOAP messages to be both human and machine readable [13]. By sharing the same features as XML described above, SOAP can accomplish information exchange between different applications which are implemented in different programming languages and running on variety of platforms. For example, a client application written in Microsoft Visual Basic could use SOAP to access a method in a CORBA object running on a Linux platform.

Secondly, utilizing HTTP as its transport protocol, SOAP eliminates firewall barriers. CORBA/IIOP and Java RMI use object methods for RPC calls which are often denied by firewalls or proxy servers because of security constraints. Therefore, CORBA/IIOP and RMI are mainly used in Local Area Network (LAN). In contrast, SOAP exploits HTTP, a standard transport protocol in the Internet, to bridge the communication between different organizations located behind firewalls [15], which makes it easier for designers to develop their distributed systems in Wide Area Network (WAN).

Finally, SOAP not only works with HTTP but also with other transport protocols such as SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol) [14], WAP (Wireless Application Protocol) [16] etc., which makes SOAP widely used in many applications and products. For instance, Liu C.C etc. [16] have set up a wireless online teaching system using SOAP; Microsoft UPnP (Universal Plug and Play) [17] uses SOAP to carry control messages between the control points and service devices.

CORBA and RMI have an advantage in keeping session information. They make use of “threads” to identify the communication between the client and the server. TCP (Transmission Control Protocol) [18], which is a connection-oriented transport protocol, acts as the keeper of the client and the server session states. Therefore, they can be easily used in developing decentralized network applications. In contrast, SOAP has its own disadvantage in this aspect. As standard HTTP is a stateless protocol [19] , SOAP naturally follows the HTTP request / response message model by providing SOAP request parameters in a HTTP request and SOAP response parameters in a HTTP response. Therefore, there is no session control on SOAP. This means that communication is only established for the duration of sending a request and receiving a response. With SOAP being widely used in Web services especially in E-commerce, a session is needed to be added to SOAP messages so as to provide stateful services, such as “shopping cart”, between participants. Unfortunately, the SOAP specification does not say anything about session management on SOAP messages. In other words, there is no standard mechanism to manage SOAP session information.

In order to keep a session state between the client and the Web server, some technologies are invited to manage the session in different methods. The following section introduces the basic concepts of session and session management and evaluates some techniques used to manage the session in HTTP, HTML and SOAP.

2.3 Session Management

2.3.1 Session and Session Management Definitions

A session is defined as "Stateful connection between two parties during which one or more communications take place" [20]. The management of a session should include the processes of starting, joining, leaving, terminating and browsing the situations of the session across the network [21].

There are two main issues that need to be solved in session oriented communication applications. One is how to manage multiple sessions on the server side; the other is how to keep the state between the client and server during communication.

Keeping session state is important for the communication in the network. In the OSI (Open Systems Interconnect) seven-layer model, TCP, which belongs to the transport layer, is a connection-oriented transport protocol. TCP provides a way to set up and manage the session between two machines through the designated ports on those machines [22]. In fact, TCP makes use of a pair of sockets to establish the connection between two machines in the network. After the sockets in both machines have set up the connection, corresponding processes are going to keep the duplex communication state between them [18].

To find a new way to manage sessions for Web services, an overview of traditional session management techniques in Web-based applications is given, and some new session management methods on SOAP messages are explored in this section.

2.3.2 Traditional session management techniques

- **Session Management in TCP/IP and HTTP**

In the IP (Internet Protocol) networking stack, HTTP belongs to the application layer and sits on top of the TCP. However, as mentioned before, HTTP is a stateless protocol, which means that the web server will not keep the client’s state or the user’s personal information. For example, when a web server receives an HTTP request, it does not know whether the request is sent by a new user or a previous user. In many E-commerce applications, keeping client’s information is an important requirement. In order to keep the session information for a user during a sequence of requests, CGI (Common Gateway Interface) and Servlet have been invented to overcome this restriction and to manage the session state for the Web users.

In the early development of the World Wide Web, the web server had to create separate processes in the OS (Operation System) to handle the requests from different browsers, with the aim of dynamically generating response pages. CGI allowed these separate processes to read data from an HTTP request and to write data to an HTTP response [23]. For every new user, CGI creates a file to store session information and an ID number that corresponds to the user of that session. This information will be written as part of every URL (Uniform Resource Locator) [24]. However these methods are platform-dependent and expensive in terms of processor and memory resources [22].

Servlet is another technology that can be utilized for dynamically constructing web pages using the HTTP request. The session management is implemented by a group of classes, “namely the Session Tracking API” [25]. When a browser sends a request to a Web server, a Servlet is invoked. It will create an HttpSession object and a cookie object. The cookie object will be transported to the browser through the HTTP header and saved on the client side. The cookie object can be used to maintain the state and to collect users’ information such as shopping cart data and so on. The HttpSession object will be saved on the server side, and used to store and retrieve application layer data associated with the client. Every cookie object has a session identifier which is linked with the HttpSession object. Servlet uses the cookie identifier to track Web users and to find the corresponding HttpSession objects on the server. In this way, Servlet can manage the session for Web applications.

Cookies, as defined by Peng, W. and Cisna, J. [26], are: "small data structures sent from a Web server to your browser and saved on your hard drive in a text file. They are nothing more than a string of characters (letters and numbers) that store certain pieces of information about you." [26] Because of simple implementation and anonymous session tracking property, the cookie technique has been widely used in practice [25]. Most browsers know that they need to retrieve the cookie name and value pair from the HTTP header and to send them back to the server during the communication between the client and the server. Powell [27] believes that using cookies is a smart decision to meet many state management requirements.

Although the cookie technique is supported by most browsers, its limitations cannot be ignored. Firstly, cookies are dangerous for clients when they shop on line. For example, when a user purchases something through the Internet, there will be a unique Web session between the browser and the Web server which is recorded by Web session ID. From the session ID and session object, the Webmaster will know what the user has purchased and his/her personal information including credit numbers. If the Website leaves cookies on the user's computer, the session ID can be found in the cookie file. Because cookies are just plain text files, they are easy to read using any text editor. Manipulating cookies is the primary way used by hackers to hijack web sessions. In addition, hackers can also alter information within a cookie file. Therefore, "a little trial and error can produce a real mess for innocent victims [28]." It can be even more dangerous if hackers use proxy servers which are an intermediary between Web server and browser. Hackers can intercept the cookie information and alter it. In fact, any hyperlink on Web pages could be connected to a hacker's proxy server.

Unfortunately, a general Web surfer cannot distinguish between the real Website and the hacker's proxy server.

Secondly, a cookie is transparent to client users, which means they do not know where the cookies come from, when they are saved on their computers and how they work in their computers. This will lead users to worry about their privacy. Therefore, some clients choose to deny cookies on their browsers. As a result, cookies lose their function of adding session state information to HTTP. A new mechanism needs to be introduced to provide cookieless session management for Servlet.

This new mechanism is URL rewriting. Instead of adding information in the HTTP header, this technique embeds a session identifier in the requested URL and transmits it back and forth between the client and the server [27]. A major drawback of this mechanism is that the identifier, which usually is a number, is directly put into page links. If the user bookmarks a page, the bookmark will include the Session ID. Therefore, if they try to revisit that bookmark later, they will get a "page not found" error because the session will have timed out. Moreover, this method may not work well for complex applications and a session ID can be easily changed and edited by hackers. Therefore, in term of security, it is not safe.

Cookies are dependant on adding sessions to HTTP layer and are allowed application layer programming languages (such as Servlet) to access its content through a defined mechanism in HTML [29]. SOAP is not solely dependent on the HTTP protocol and is not a HTML file. This means that the cookie technique is not suitable for applying to SOAP session management. Similarly, URL rewriting gives each user a specific URL for "talking" to the web server, but this URL information cannot be transmitted to the SOAP messages to make the SOAP store the session information [29].

- **Session Management in HTML**

A simple approach of adding session management in HTML is hiding session information in HTML forms. When an online user surfs a Web page and fills a form on that page, the session ID will be hidden in the form and be transported between the client and the server. This session ID will be easily found by search the source code of the Web page. Because it is an HTML session specification, it cannot be used for managing a SOAP session, as there is no HTML form in SOAP messages.

2.3.3 Session Management on SOAP Messages

SOAP provides a much more powerful interface for Web services and allows users to make more complex method calls between different Web-based applications running on various platforms. Therefore, many researchers and Web developers want to take advantage of these features and use SOAP to enhance their Web application functions.

However, as mentioned above traditional session management techniques cannot support SOAP session management. In order to use SOAP in Web services, two main techniques are designed for adding session information to SOAP calls.

- **Integration of session management into Web servers**

In Tsenov's [14] project named REGNET, Zap software is used to manage the session for SOAP messages. The project's aim is to "set up a functional Network of Cultural Service Centers through Europe" [14]. They presented a solution for applying SOAP protocol in E-commerce. As introduced by Tsenov, Zap is a software module which belongs to the Apache Web server. It can hide complex session management. In the project, the Apache Web server is used to provide HTTP service for SOAP calls.

The integration of session service into the web server has its advantages and disadvantages. One advantage is that it is easy for operators to use the software and to develop the whole system. The other one is that it alleviates the burden of maintaining the system and enhances the efficiency of the system [14].

Although, this method is easy for the developer and maintainer of the system, one disadvantage cannot be ignored. Zap is a part of the Apache server, it may not cooperate with other Web servers for session management. In other words, compatibility with other Web servers is a problem for Zap. Furthermore, the session management and the HTTP servers are bound tightly, thus the session information will be lost when the server crashes. From the programmers' point of view, the session management is like a black box. If the Web server crashes, they can do nothing to recover the session information for the Web users. If this happens, it would undermine Web users' confidence in E-commerce.

- **Adding state context in the SOAP header**

Another solution for adding session management on SOAP messages is by making use of the SOAP header to provide stateful services. In Jeckle's research [20] on SOAP 1.2, the SOAP header has four features:

1. The SOAP header is basically raw information to SOAP messages
2. SOAP users can define the header by themselves
3. The SOAP header has its own namespace which is different from the SOAP namespace.
4. The SOAP nodes, the intermediaries between the SOAP sender and the ultimate receiver that can process parts of the transported information along the SOAP message path, can deal with SOAP header information.

Meanwhile, a SOAP node, as describe in Jeckle's presentation, has three other corresponding attributes:

1. A SOAP node can process SOAP messages along their message path.
2. A SOAP node can split a SOAP header when it matches the actor defined in the SOAP header.
3. A SOAP node can be located by a SOAP header using URI (Uniform Resource

Identifier).

Based on these characteristics, Jeckle defined the context information in the SOAP header. The items in the header include context ID, expire time, issuer and so on. Jeckle asserted that this solution has infrastructure and programming language independence, because it uses SOAP’s built-in extensibility mechanisms. By using this solution, SOAP will not depend solely on HTTP to transport its messages, but also can use other transport protocols, such as SMTP. In addition, SOAP can have more chance to cooperate with many other programming languages besides Java. However, Jeckle left the session management implementation to the Web developers. This means that each developer has to choose his/her own mechanism to manage the SOAP session information.

Microsoft Corporation also provides this solution for supporting the SOAP session. They define the DSML (Directory Services Markup Language) SOAP session namespace as follows:

xmlns=”urn:schema-microsoft-com:activedirectory:dsmlv2”

Three SOAP header elements, <BeginSession> <Session> <EndSession>, and their specifications are also presented for operating SOAP sessions [30].

However, from the specification it can be seen that this version of a SOAP session is still based on the cookie mechanism. It puts cookies in the <Session> tag, which is included in the SOAP header [29]. Also, the <Session> element will be transmitted between the client and the server just like HTTP cookies. The developers have to make sure the client application treats the <Session> tags as HTTP cookies and sends them back to the server with each request. Furthermore, this mechanism assumes that SOAP is transported on HTTP which limits the cooperation between SOAP and other transport protocols.

Based on the limitations of the existing solutions, a new way of adding session information for SOAP calls is needed. Session Initiation Protocol, which is a standard session management protocol, is a method to accomplish this goal.

2.4 Session Initiation Protocol (SIP)

2.4.1 The Definition and Functions of SIP

SIP (Session Initiation Protocol) is developed by the Internet Engineering Task Force (IETF). It is a standard application-layer control protocol for setting up, modifying and terminating sessions regardless of media content [31]. The major applications of SIP are Internet conferencing, Internet phone, Internet games and online chatting.

There are five main functions that SIP provides for operating the communication between two end users. Firstly, it can find the called user location in the communication. Secondly, SIP is able to determine the availability of the end user. Thirdly, the communication media and its parameters will be discovered by SIP and fourthly, another function sets up the session parameters for both called and calling parties. Finally, SIP can manage the session "including transfer and termination of sessions, modifying session parameters, and invoking services" [32].

2.4.2 SIP Protocol Stack

Figure 5 illustrates the protocol stack of SIP. SIP is an application layer protocol which usually carries SDP in its body and utilizes TCP or UDP (User Datagram Protocol) [33] to transfer its messages through the Internet. In the latest SIP specification RFC 3261 [32], TLS (Transmission Layer Security) version 1.0 [34] has been added to be another transport protocol for SIP. This means that SIP can use TLS over TCP for encrypted transport with the additional capabilities of authentication [35]. The complete forms for some of the abbreviations in Figure 5 are shown below:

- SDP : Session Description Protocol [36]
- SAP : Session Announcement Protocol [37]

- RTP : Real-Time Transport Protocol [38]
- RTSP: Real-Time Streaming Protocol [39]
- IP : Internet Protocol [40]

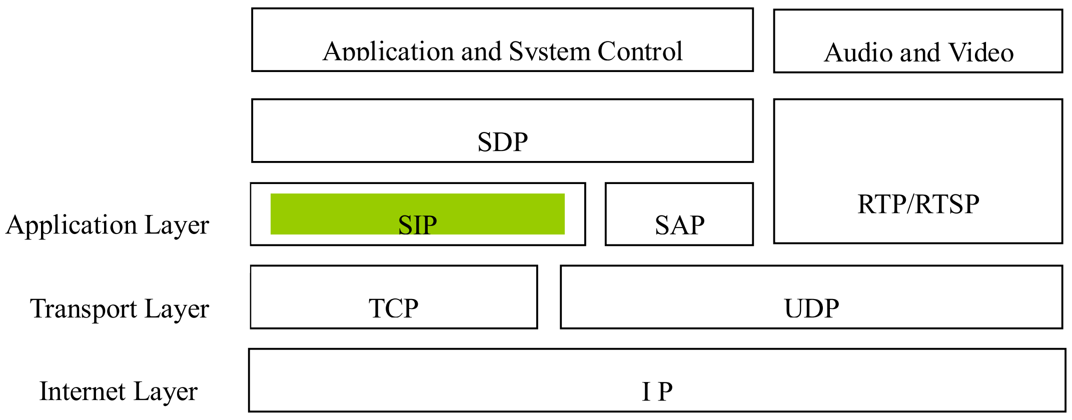


Figure 5 SIP Protocol Stack (From [41])

2.4.3 SIP Four Components

SIP consists of four major components: SIP User Agents, SIP Registrar Servers, SIP Proxy Servers, and SIP Redirect Servers. Firstly, SIP User Agents (UAs) can be divided into UAS (User Agent Server) and UAC (User Agent Client). Usually, these are end-user devices, such as PCs, telephones. Secondly, SIP Registrar Servers are databases which contain the UAs addresses within one domain. Thirdly, SIP Proxy Servers accept a session request from a UA, look up the callee address in SIP Registrar Servers, and then send the inviting session request directly to the callee if the callee is in the same domain. If not, the Proxy Servers will send the request to another SIP Proxy Server. Finally, SIP Redirect Servers allow the SIP proxy Server to allocate the callee address in a different domain [32].

2.4.4 SIP Messages

SIP is a text-based protocol, which means that the SIP messages can be read directly by a user and are easier to extend with new features [42]. Actually, SIP is based on

HTTP-like request and response model, so its messages can be divided into SIP Request and SIP Response [32].

- **SIP Request Messages**

A SIP request consists of a request line, several messages headers, an empty line and a messages body. Figure 6 shows the format of a SIP request.

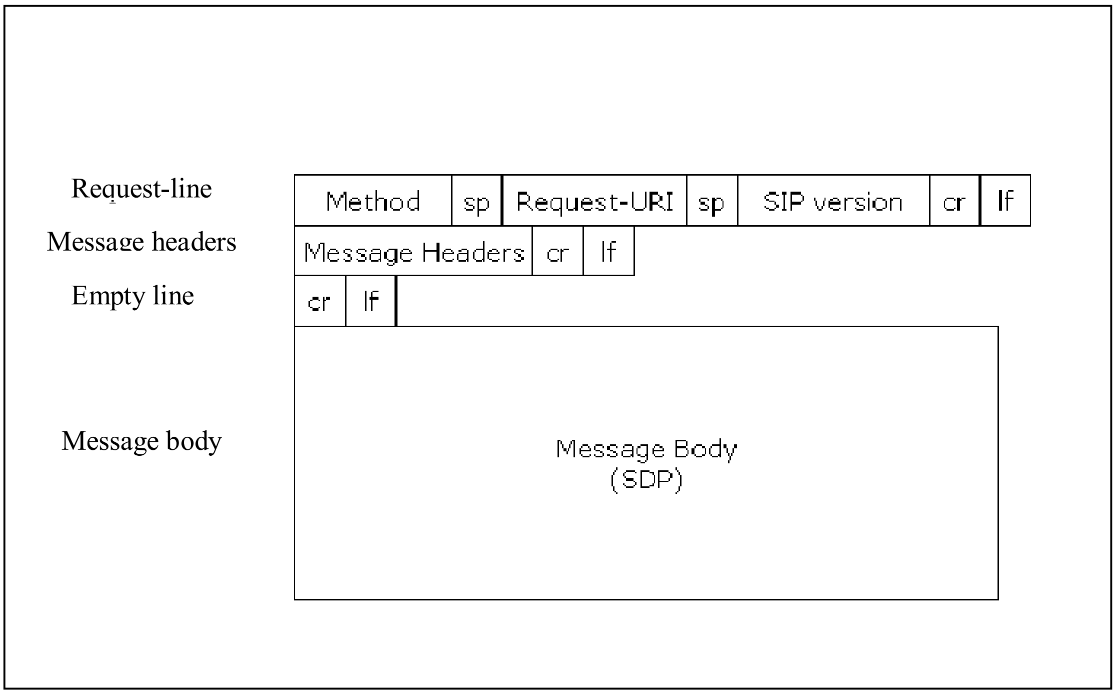


Figure 6 SIP Request Message Format (From [41])

Basically, there are six methods that can be presented in the request line of SIP request messages. The methods and their functions are listed in Table 1.

SIP methods	Function description
INVITE	Establish media session between user agents
REGISTER	Register a user agent current contact IP address
BYE	Terminate an established media session
ACK	Acknowledge final responses INVITE requests
CANCEL	Terminate pending searches or call attempts
OPTIONS	Query a user agent’s capability and discover its current availability

Table 1 SIP Six Methods and Their Functions (From [35])

- **SIP Response Messages**

A SIP response consists of a status line, several messages headers, an empty line, and a message body. Figure 7 shows the format of a SIP response.

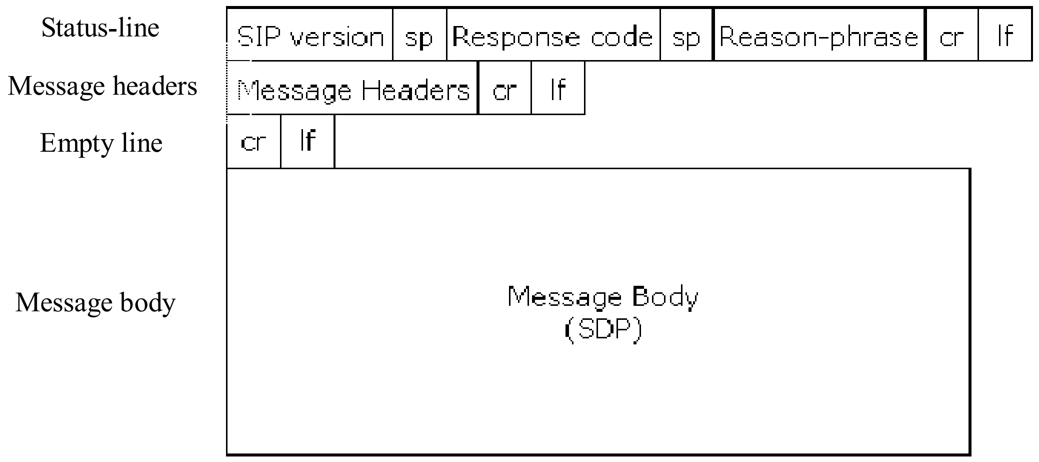


Figure 7 SIP Response Message Format (From [41])

A SIP response message is generated by a SIP user agent server or a SIP server to reply a user agent client request. There are six classes of SIP responses which can be presented in the status line of the SIP response messages. The first five classes are the same as HTTP; the last one is created for SIP. Table 2 lists the six classes and their descriptions and actions.

Class	Description	Action
1xx	Informational	Indicates status of call prior to completion. If first informational or provisional response.
2xx	Success	Request has succeeded. If for an INVITE, ACK should be sent; otherwise, stop retransmissions of request.
3xx	Redirection	Server has returned possible locations. The client should retry request at another server
4xx	Client error	The request has failed due to an error by the client. The client may retry the request if reformulated according to response.
5xx	Server failure	The request has failed due to an error by the server. The request may be retried at another server.
6xx	Global failure	The request has failed. The request should not be tried again at this or other servers.

Table 2 SIP Response Classes (From [35])

- **SIP Message Headers**

SIP message headers are presented in both SIP request and response messages. The commonly used headers are CALL-ID, CSEQ, FROM, TO, VIA, CONTENT-TYPE, and CONTENT-Length [41]. These header fields and their descriptions are shown in Table 3.

Headers	Descriptions
CALL-ID	Uniquely identify a call between two user agents
CSEQ	Contain a decimal number which increase for each request
FROM	Indicate the originator of the request
TO	Indicate the recipient of the request
VIA	Record the SIP request and response routes
CONTENT-TYPE	Specify the Internet media type in the SIP message body
CONTENT-Length	Indicate the number of octets in the SIP message body

Table 3 SIP Message Headers (From [35])

- **A Simple SIP Session Establish Example**

To make things concretely, a simple SIP session establish example is given below. (Shown in Figure 8)

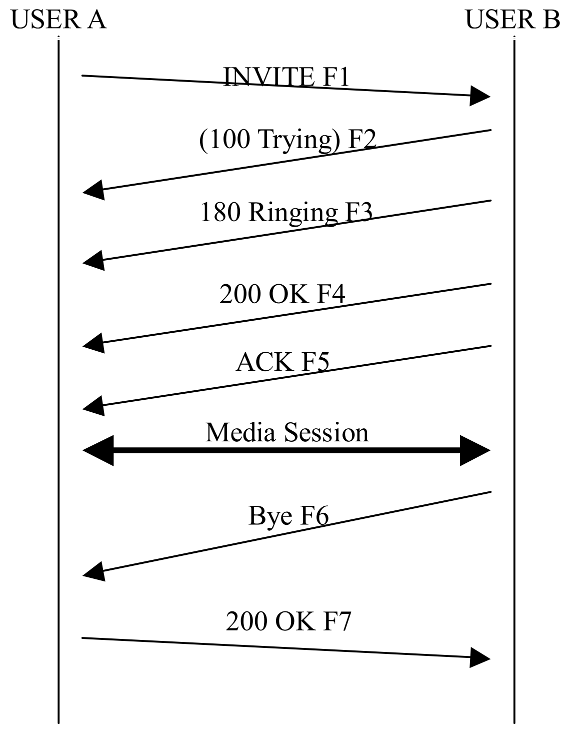


Figure 8 Standard Peer To Peer SIP Session Sequential Diagram

(F1 - F7 are SIP messages, From [31])

The processes of setting up and terminating a session between two peers consist of six steps which are showed in Figure 8. Initially, User A sends an INVITE message to User B to initiate a session; then User B responds to User A by sending a 100 Trying message, a 180 Ringing message and a 200 Ok message and tells User A that it agrees to set up the session with User A. After User A receives these messages, it sends back an ACK message and shakes hands with User B. A session now is set up and the information can be transported back and forth between two peers. When User B wants to finish the session, it sends a BYE message to User A. User A will reply with a 200 OK message to the other side and the session is finished.

In this example, User A can be regarded as a UAC whereas User B can be regarded as a UAS. When a UAS receives an INVITE message, it needs to know what kinds of media session the UAC wants to set up. At the same time, UAS also needs to know how to distinguish different invitations from UAC when it gives a response. All these jobs are done by reading the SIP message body information, which is a SDP message.

2.5 Session Description Protocol (SDP)

2.5.1 SDP Definition and Fields

SDP is developed by the IETF Multiparty Multimedia Session Control (MMUSIC) working group [36]. Much like SIP, SDP is also a text-based protocol, which is used to describe multimedia sessions for session initiation, session invitation and session announcement [36]. It can be carried in Session Announcement Protocol (SAP), Session Initiation Protocol (SIP), Real-Time Streaming Protocol (RTSP), electronic mail using the MIME (Multipurpose Internet Mail Extensions) extensions, and the Hypertext Transport Protocol (HTTP) [36]. Included in the SDP messages are session names and purposes, the active time of the session, the media included in the session, and receivers' information such as addresses, ports, formats and so on [36]. All the SDP fields are listed in their required order in Table 4.

Field	Description	Mandatory
v=	Protocol version	Yes
o=	Owner/creator and session identifier	Yes
s=	Session name	Yes
i=	Session information	No
u=	URI of description	No
e=	Email address	No
p=	Phone number	No
c=	Connection information	No
b=	Bandwidth information	No
z=	Time zone adjustments	No
k=	Encryption key	No
a=	Attribute lines	No
t=	Time the session is active	Yes
r=	Zero or more repeat times	No
m=	Media information	Yes
a=	Media attributes	No

Table 4 SDP Field List in Their Required Order (From [36])

2.5.2 A Simple Example of SDP message

A SDP message example is shown below. (See Figure 9)

```
v=0

o=John 20040607111144 20040607111144 IN IP4 43.32.1.5

s=SDP Message

i=A SDP Example

c=IN IP4 225.45.3.56

t=2877631875 2879633673

m=video 23422 RTP/AVP 31
```

Figure 9 A SDP Message Example

There are two important fields in Figure 9 that need to be explained here. The first one is "o=". It contains the session creator name "John", and the session identifier "20040607111144", thus a SIP UAS can use this field to identify the session. Another important field is “m=”, which indicates the media type “video”, port number “23422”, transmit protocol “RTP/AVP”, and the media format presenting in number style “31”. By reading this field, a SIP UAS will know the media types that a UAC wants to communicate with and which kinds of transport protocols are used in the session.

2.6 Summary

In this chapter, Web service and the technologies behind, i.e. UDDI, WSDL and SOAP were introduced. By comparing these with other distributed technologies such as CORBA and RMI, the advantages and disadvantages of using SOAP in Web services have been made obvious. SOAP can easily integrate Web-based applications across different platforms and programming languages in WAN. However, there is no standard way of managing the session on SOAP messages. Following that, current session management techniques on HTTP, HTML and SOAP were reviewed. After analyzing these session management techniques, it was believed that the techniques based on HTTP and HTML to manage the session cannot be applied on SOAP messages, and the current SOAP session management methods have their limitations. Thus, a new method needs to be developed to manage SOAP sessions. Finally, SIP, a standard session management protocol, and SDP, which can be used to describe a session and is carried by SIP messages, are introduced. The aims of this thesis are firstly to use SIP to add session management to SOAP messages, and secondly to do the transaction management on Web services in an advanced stage. The next chapter will present background on transaction management and propose the aims of this thesis.

Chapter 3 Web Service Session and Transaction Managements Using SIP

3.1 Introduction

Some currently used session management techniques for Web-based applications and SIP and SDP have already been introduced in Chapter 2. By using SOAP for information exchanging, Web services technology enables companies to integrate applications from disparate platforms and programming languages into a composite application [43].

The primary objectives of this thesis are to propose and develop a new mechanism to support session management on Web services, and then based on this mechanism, to accomplish Web services transaction management. In this chapter, an overview of the transaction management is described in section 3.2. Section 3.3 provides the hypotheses of this research. After that, the implementation gaps of Web services session and transaction managements will be discussed in section 3.4. Then, the aim of this thesis is presented in section 3.5. Finally, section 3.6 provides the summary of this chapter.

3.2 Overview of Transaction Management

3.2.1 ACID Transactions

ACID is an acronym for Atomic, Consistent, Isolation, and Durable [44]. The concept of ACID transactions has come from database transaction and is widely used in distributed database environment.

ACID transactions identify a logical unit of work that is either completely accomplished or done nothing at all [44]. In other words, the actions in a transaction either all happen or are all undone. The four attributes of an ACID transaction are: atomicity, consistency, isolation, and durability [45].

- Atomicity: The transaction is atomic. That is either all the actions occur or none of them occur. If the transaction is interrupted and cannot be completed for any reason, the state should be recovered as it was before the transaction happened [45].
- Consistency: The transaction should keep the data in a consistent state. An inconsistent state might happen during the transaction, but no other transaction can access this inconsistent data at this time. When the transaction is finished, all the inconsistent data will be eliminated [45].
- Isolation: Transactions are isolated from each other. That is, the effects of concurrent transactions are not visible until one of the transactions is committed [45].
- Durability: Once a transaction is committed, the effects of it should be guaranteed to survive even if the system crashes [45].

The ACID transactions give a simple transaction management model which is referenced by database programmers. When transactions happen simultaneously in a distributed database environment, abiding by the ACID transactions principle is most important.

3.2.2 Web Services Transaction Management

The ACID transactions have given a clear and simple model for programmers to control transactions in a distributed environment. However, these require joined applications and components to talk to each other under a highly coupled environment [46]. Web services are usually loosely coupled applications, and not all business applications can be strictly implemented ACID transactions. Thus, Web service based transaction management needs to be standardized.

In fact, transaction management on Web services is a new issue that is currently being addressed by industry leaders [47]. The Arjuna Technologies, Fujitsu Limited, IONA Technologies, Oracle Corporation, and Sun Microsystems have proposed the Web Services Composite Application Framework (WS-CAF) specification [48]. Three parts of specifications are included in WS-CAF specification: Web Services Context Service Specification (WS-CTX) [49], Web Services Coordination Framework Specification (WS-CF) [50], and Web Services Transaction Management Specification (WS-TXM) [51]. The relationship between these three specifications and transaction protocols is show in Figure 10.

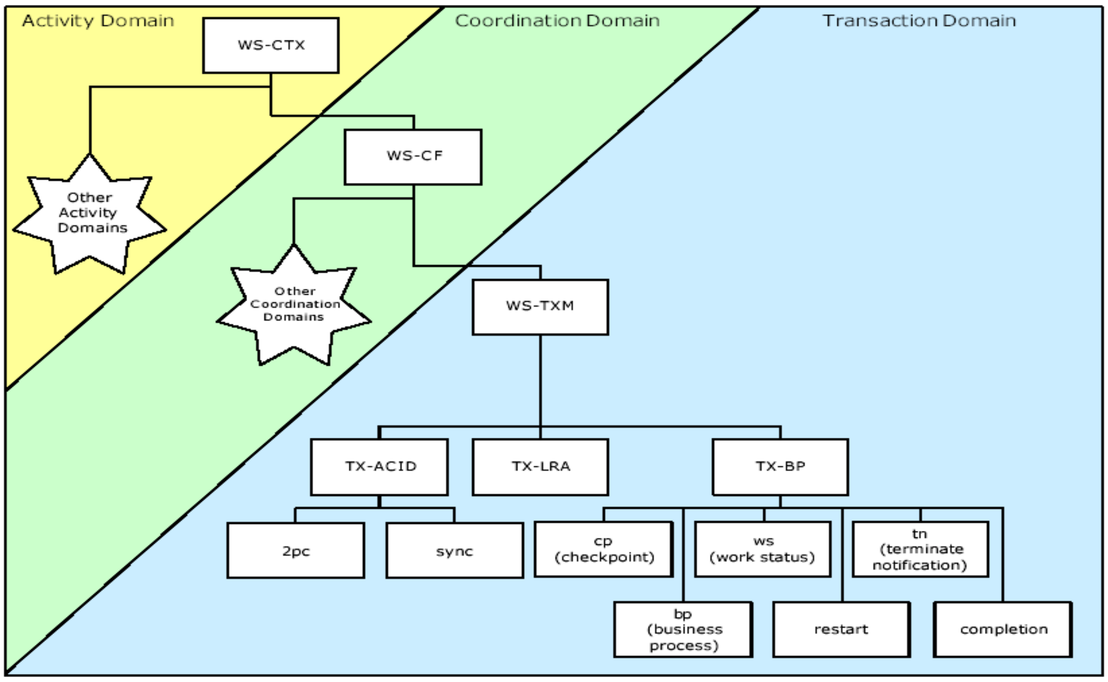


Figure 10 Relationship between Specifications and Transaction Protocols (From [48])

From Figure 10, it can be seen that an implementation of the WS-CAF consists of three steps. Firstly, a context service needs to be set up to provide context management. Secondly, a Web services coordination framework can be build on the context service to supply the message delivery function. Finally, a Web services transaction management system can be added on the Web services coordination framework to realize a variety of transaction management types.

The three Web transaction management models proposed by Bunting D. et al in the Web Services Transaction Management (WS-TXM) specification [51] are the ACID transactions model, the long running action model, and the business process transaction model.

- ***ACID transactions model (TX-ACID)*** is the traditional ACID transaction and can be used to interoperate the existing transaction infrastructures [51].
- ***Long running action model (TX-LRA)*** is based on the "all or nothing" atomicity attribute of ACID transactions. The difference is that the activities in the long running action model are not compulsory to perform all ACID transaction requirements [51].
- ***Business process transaction model (TX-BP)*** is based on the business specification to manage transactions. A business process transaction may consist of ACID transactions or long running actions [51].

In general, ACID transactions are still the cornerstone in these three transaction management models. However, because Web services can be applied to different kinds of businesses, in some circumstances, such as long-lived applications, the atomicity and isolation attributes cannot be strictly implemented according to the ACID transactions.

In addition, the newest movement on Web service transaction management made by W3C was the development of Web Services Choreography Description Language (WS-CDL) [52]. The WS-CDL is an XML-based language which is independent of business processes, allowing an interoperable framework between different platforms and programming languages [52].

To sum up, the WS-CAF and WS-CDL just give us a guideline to implement Web service transaction management. According to Siegrist J.L. [47], Web services are a relatively new technology, and many issues, such as transaction management, security, interoperability etc, are needed to be considered. And the solutions of these problems are new and not complete.

3.3 Hypotheses

Based on the literature reviewed in the above section and the previous chapter, a number of hypotheses are generated for investigation.

Hypothesis 1: The standard session control protocol — SIP can be used to manage sessions and transactions for Web services.

Hypothesis 2: Using SIP to manage sessions for Web services is more elegant and general than Jeckle's and Microsoft Corporation's solutions.

Hypothesis 3: Using SIP to manage transactions for Web services can keep the atomicity and consistency of the transactions.

Hypothesis 4: Using SIP to manage sessions and transactions for Web services provides an easier solution on security issue than other solutions proposed so far.

3.4 Web Services Session and Transaction Management Implementation Gaps

3.4.1 Session Management Implementation Gaps

There are three gaps in implementing session Management on Web services. They are session description, the session information extraction from SOAP messages for Web services and the session management mechanism based on Web services architecture.

As mentioned in chapter 2, some mechanisms have been used in adding session information for SOAP messages such as making use of HTTP cookies or by adding session information in the SOAP header. The latter is recommended by W3C [8] and is also adopted in this thesis. However, the session information inserted in the SOAP header block varies. If two different Web services adopt same mechanism to add a session identifier (ID) for their services, different customers, who belong to these two Web services, might have the same session ID. When these two Web services communicate with each other, the phenomenon of different sessions using the same session identifier will occur. Thus, finding a standard way to uniquely identify a session in cyberspace is the first issue in managing the Web service sessions.

To allow the Web services to identify the session information contained in SOAP messages is the second implementation gap for session management. Traditional Web servers are usually HTTP servers, such as Apache Tomcat, Microsoft IIS etc, which can interpret HTTP requests and give HTTP responses. The bodies of the HTTP requests can be HTML pages, Java Server Pages (JSP), Active Server Pages (ASP) and so on. However, these kinds of Web servers cannot understand SOAP messages unless they are equipped with a SOAP interpreter or equipped with a SOAP server. Once a HTTP server has the ability to understand SOAP requests, it can recognize methods and parameters in SOAP body messages, invoke the back end Web services programs and give responses to them. This procedure simply follows the HTTP request and response style which means that the Web services programs have no opportunity to extract the session information included in SOAP headers. Therefore, the second issue of managing Web service sessions that needs to be considered is how to deliver the session information from a SOAP server to back end Web applications.

There is no formal definition on adding session management for SOAP messages, as Newmarch J. has pointed out [29], “The current state of Web services with regard to session management is: ‘roll your own’”. This means that the Web services programmers have to choose their own ways to develop the session management mechanism for tracking the Web services session state. If every Web service has its own way to manage the session, it will lead to an interoperation problem between Web services. For instance, if a user wants to buy a book online, and then rent a video online and finally pay, it is impossible to manage this consumer behavior if the book shop Web service and the video shop Web service use different session management mechanisms. Hence, developing a new session management mechanism which utilizes a widely accepted session management protocol and which is based on Web service architecture is the third issue to be considered in this thesis.

3.4.2 Transaction Management Implementation Gaps

As described in Section 3.2, there is no complete solution for Web service transaction management, and currently some industry leaders just provide some implementation guideline. As a result, solutions to Web service transaction management are different from business to business, depending on needs and complexity. Hence, a Web service transaction management solution, which is simple to be implemented, easy to be extended by adding other properties such as security and interoperability, is in urgent demand. These are also the gaps for implementing Web service transaction management.

3.5 The Aims of This Thesis

In order to fill in the gaps mentioned above, this thesis aims to develop a new session management mechanism for Web services technology by using a standard session control protocol SIP and then, based on this mechanism, to provide a simple Web service transaction management solution and implement it in an E-banking Web Services System.

3.5.1 Two-Level Implementation

- Web Services Session Management Implementation

  For implementing the Web services session management mechanism, an Online

Video Shopping Web Services System has been developed. Apache AXIS (Apache eXtensible Interaction System) was chosen to act as SOAP server Tomcat, which included in JWSDP 1.2 (Java^{TM} Web Services Developer Pack), was the HTTP server. The client application for the Video Shopping System was a Java application. The SIP server and the SIP client were developed in Java. Microsoft Access was used to store the video data and customer information.

In the first level implementation, a Web service called "VideoService" which provides online video shopping service for its customers was set up. A SIP server was used to manage sessions between the client application and the "VideoService". A shopping cart, which associates with the session between the client and the Web service, was created for every client and was used to record the videos selected by the client.

- Web Services Transaction Management Implementation

Based on the first level implementation, an E-banking Web Services System was developed which provided money transferring service from Bank A to Bank B. In this system, two bank Web services were developed for supplying the money transferal service. A "Transaction Manager" was introduced to manage the transaction of transferring money. In fact, the transaction manager was also a Web service, which was able to communicate with SIP server to get the session information of the E-banking clients and to manage the money transferal transactions for the clients.

In this system, the SOAP server and HTTP server were the same as the Video Shopping System. The SIP server, the SIP client and the system client were all developed in Java. Microsoft Access was chosen to store the bank account data.

3.6 Summary

This chapter first introduced traditional ACID transactions and described the current implementation state of Web services transaction management. Although industry leaders have proposed some guidelines for implementing the transaction management for Web services, a standard transaction management mechanism is still under development. After proposed the research hypotheses, the Web services session and transaction management gaps were listed. With the aim of addressing these gaps, this chapter presented the objectives of this thesis and gave a brief introduction of the following two systems — the Online Video Shopping Web Services System and the E-banking Web Services System, which were implemented in this thesis. The next two chapters will present the analysis, design and implementation of these systems in detail.

Chapter 4 Session Management on Web services by Using SIP

4.1 Introduction

In the previous chapter the current implementation state of Web service transaction management and the gaps in implementing Web service session and transaction management were reviewed.

In this thesis, two Web service based systems—an Online Video Shopping Web Services System and an E-banking Web Services System were developed to accomplish the session and transaction management using SIP.

The first step to accomplish Web services transaction management is to provide a session or context control for the Web services. In other words, the realization of the Online Video Shopping Web Services System is a prior condition of the E-banking Web Services System performance. Hence, this chapter focuses on the implementation of Web services session management by taking the Online Video Shopping Web Services System as the sample project. Based on the session management mechanism used in the first project, the implementation of Web services transaction management will be explained in the next chapter. The E-banking Web Services System will be the example to demonstrate the implementation.

In this chapter, the system analysis is provided in section 4.2. Section 4.3 presents the design of the Online Video Shopping Web Services System. The system deployment is illustrated in section 4.4. This is followed by a system implementation discussion in section 4.5. Finally, session 2.6 provides the summary of this chapter.

4.2 System Analysis

4.2.1 Introduction of the Online Video Shopping Web Services System

The Online Video Shopping Web Services System simulates a video store that sells videos through the Internet. There are six functions provided by the video store Web service and they are listed as below:

- ***List video category*** displays all the video categories in the store. The videos in the shop are classified into different catalogues such as comedy, horror, tragedy and drama, etc. It is convenient for customers (system clients) to look up their favorite videos according to these categories.
- ***List Video*** lists all the videos for a given category. For example, if a user wants to search a comedy video, he or she can choose the comedy category and then require the system to list all the comedy videos.
- ***Add to Shopping Cart*** adds a given video item to the "shopping cart". After a customer selects a video, he or she can use this function to add the video item to the "shopping cart". Hence, this function offers a "shopping cart" for the customer to keep the items that have been selected.
- ***Delete from Shopping Cart*** deletes a given video item from the "shopping cart". If a customer does not want to buy one or more video items in the current "shopping cart", he or she can use the delete from shopping cart function to "take off" the videos. Therefore, the function is to allow customers to change their mind and to delete one or more video items from their "shopping cart".
- ***Get Video Items*** enables customers to review all the items in their current "shopping cart". After selecting some videos, a customer might need to have a look at what is in the "shopping cart" and check whether they are videos that he or she wanted.

- ***Purchase*** sends the video order to the service provider (the video store). After a customer selects all the videos, he or she decides to send the order to the video store. The purchase function is used to input personal information and to buy the videos in the "shopping cart".

It is assumed that the client application knows where the video store Web service is. Hence, the UDDI's service registry role is not included in the system. The aim of this project is to accomplish session management on Web services by using SIP. The following parts of this section are organized as follows: firstly, a basic SOAP system diagram is shown to illustrate the entities involved in a SOAP based Web service system. Secondly, by analyzing the SIP session establishing process within one domain, the useful SIP components for controlling the Web services session will be established. Finally, the architecture of the Video Shopping Web Services System will be given.

4.2.2 Basic SOAP System Entities

There are four entities involving in a basic SOAP system: a SOAP Client, a Web server, a SOAP server and Web Services. (See Figure 11)

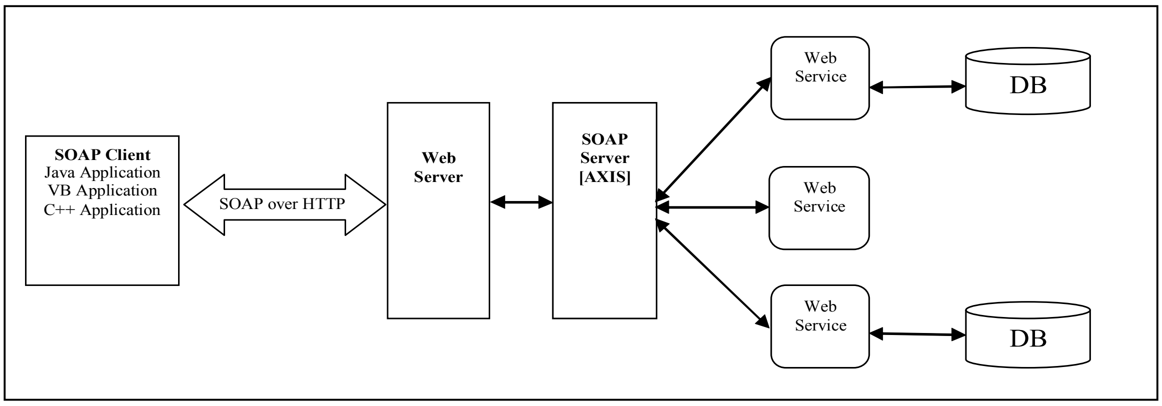


Figure 11 A Basic SOAP System Architecture (From [3])

The functions of these entities are explained below:

- ***A SOAP Client*** sends SOAP requests and receives SOAP responses over HTTP. The SOAP client can be an application which is developed in Java, Visual Basic or C++.
- ***A Web Server*** receives SOAP requests and gives responses to a SOAP client. In order to parse both the HTTP header and SOAP messages sent from clients, a Web server needs to contain a SOAP server.
- ***A SOAP Server (or a SOAP Engine)*** parses SOAP request messages, invokes the appropriate Web service, and generates SOAP response messages to SOAP clients. AXIS is version 3 of Apache SOAP, in which Java is used to implement standard SOAP. Axis is the SOAP server to be used in the system.
- ***Web Services*** provide services for the clients to use. The functions of the services are based on the business requirements. Web services can have back end database program support for providing more powerful services. Web service programs can be developed in Java, C++, and Visual Basic etc.

4.2.3 SIP Session Establishing Process within One Domain

In chapter 2, we have introduced four components of SIP: User Agents (UAs) (User Agent Client and User Agent Server), Registrar Servers, Proxy Servers, and Redirect Servers. In fact, not all these components need to be presented within one domain. For example, SIP Redirect Servers are only used when a user agent wants to set up a session with another user agent that is located in a different domain. However, to simplify the procedure this situation is not considered at this stage. Figure 12 shows the session establishing processes within the same domain.

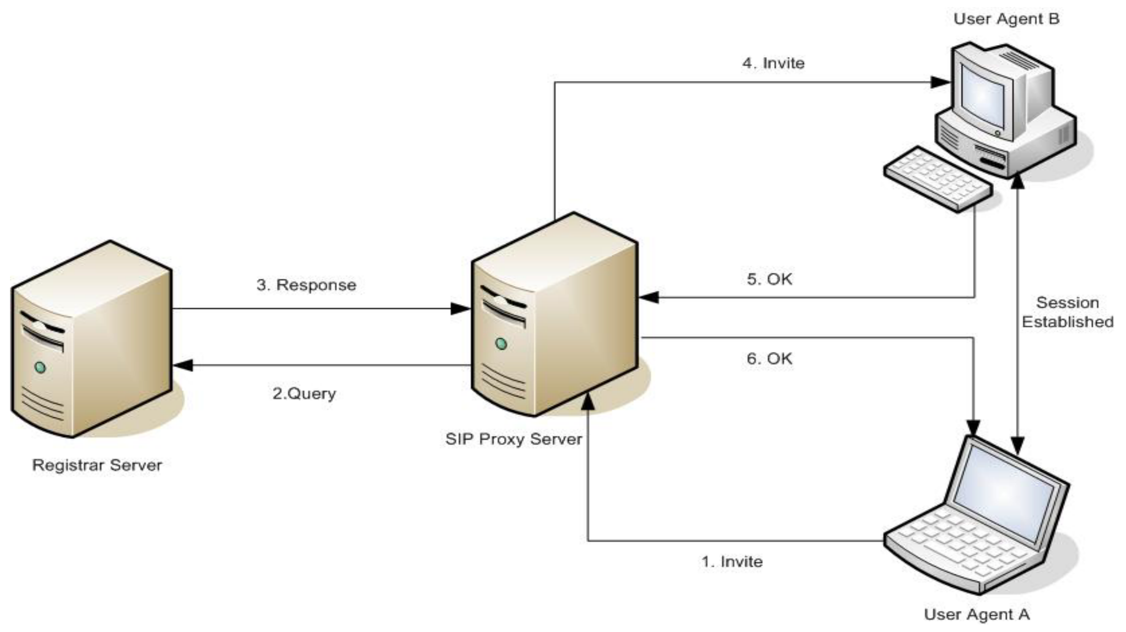


Figure 12 SIP Session Establishing Process within One Domain (From [32])

The process of setting up a SIP session within one domain is: first, User Agent A and User Agent B register their IP addresses with the SIP Registrar Server. Then User Agent A sends an inviting call to the SIP Proxy Server and tells the proxy that it wants to communicate with User Agent B. After that, the SIP Proxy Server will query User Agent B's address information in the SIP Registrar Server and the SIP Registrar Server will respond with the address of User Agent B. Then the SIP Proxy Server will send the session invitation to User Agent B. User Agent B will respond to the SIP Proxy Server to inform that it is ready to communicate with User Agent A. Then the SIP Proxy Server passes this information to User Agent A and the SIP session is established. User Agent A and User Agent B can communicate with each other through the session[32].

SIP is widely used in the Internet phone and Internet conferencing areas, such as VoIP (Voice over IP) applications, which allow voice and data to transfer over the Internet [41]. In these applications, the addresses of the User Agents are important information that the applications need to keep and update, because these addresses are the key clue for finding and sending phone calls to User Agents. The registrar server is responsible for receiving registration requests from UAs and for providing the addresses of UAs when it receives a proxy server's address query. However, in a Web service system, a Web server is not concerned about its clients IP addresses. Every client can just surf the service without any operation. Under these circumstances, there is no need to keep all clients IP addresses. Thus, the registrar server is not useful for controlling the Web services session in this system.

If the registrar server is excluded from the system architecture, there is no reason for keeping the proxy server in the system because the main function of the proxy server within one domain is accessing the registrar server and helping to decide where the callee party is. The proxy server does not issue requests or parse SIP message bodies [35]. Therefore, there is also no need to maintain the proxy server in this system architecture.

Until now, the remaining components in Figure 12 are User Agent A and User Agent B. Actually, the User Agent A can be regarded as a SIP User Agent Client (UAC) and User Agent B can be regarded as a SIP User Agent Server (UAS). If the UAC knows the UAS address, the session establishing process will be as simple as setting up a session between two peers, which has been introduced in section 2.4 of chapter 2. Based on the analysis of above, the SIP components used in the Online Video Shopping Web Services System are a SIP UAS and a SIP UAC. It is assumed that the UAC will already know the UAS IP address regardless of whether they are located within the same domain or not.

4.2.4 Architecture

As shown in Figure 13, a Web service user can establish a session with the system and then invoke Web services provided by a service provider. After the user finishes all activities with the Web services, he or she can terminate the established session.

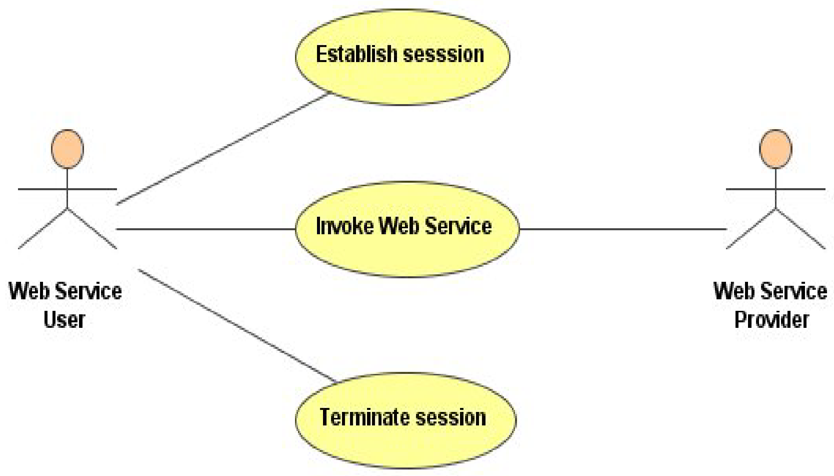


Figure 13 Use Case Diagram for Session Management on Web Services

Figure 14 illustrates the architecture of the Online Video Shopping Web Services System. On the client side, there is a client application which allows users to set up a session with a SIP server and to access the video shopping Web services. The SIP client included in the client application can be regarded as a SIP UAC, and the SIP server can be regarded as a SIP UAS. The latter is used to manage the session between the client application and the Web services. The Web service server contains a HTTP server, an AXIS SOAP server, a database and the Web services published by the service provider.

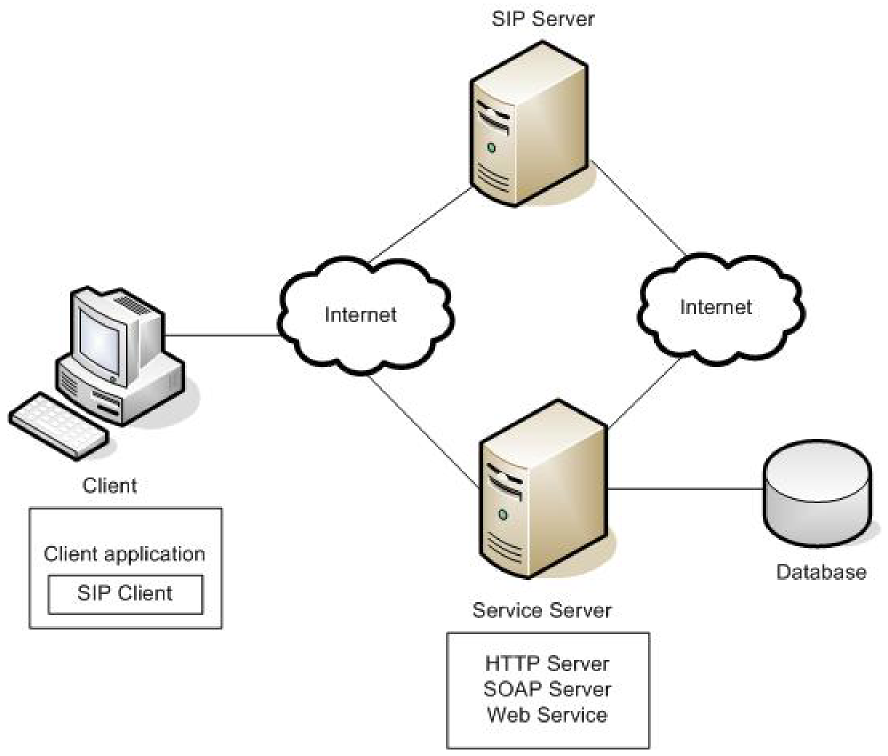


Figure 14 Architecture of the Online Video Shopping Web Services System

4.2.5 Session Establishing

With the purpose of using SIP to manage the session for Web services, a SIP server needs to be running before a user logs on the client application. The SIP server opens a server socket to accept the SIP client messages. The process of session establishing is shown in Figure 15. When the Web service user starts the client application and logs onto the system, an invitation message is sent from a SIP client to the SIP server. Then the SIP server will send a 200 OK message to the SIP client. Next the SIP client sends an acknowledge message to the SIP server and the session is established between the client application and the SIP server. Now the client application has got an unique session identification (ID), which is also recorded by the SIP server.

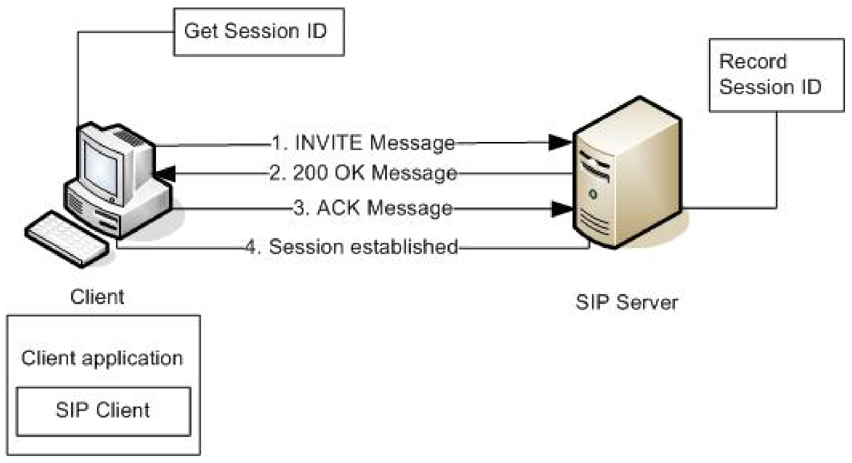


Figure 15 Session establishing between the Client and the SIP server

4.2.6 Web Services Invoking

As illustrated in Figure 16, a user can access all the services offered by the service provider, after he or she gets the session ID. The session ID will be inserted into the SOAP header element and carried by every SOAP message which is transmitted between the client application and the service server. When the service server receives a user's SOAP request which contains a new session ID, it will check that session ID with the SIP server and to find out whether the user has already set up a session. If so, the service server will first recode the session ID and then create a "shopping cart" for the user; and finally send a corresponding SOAP response message to the client.

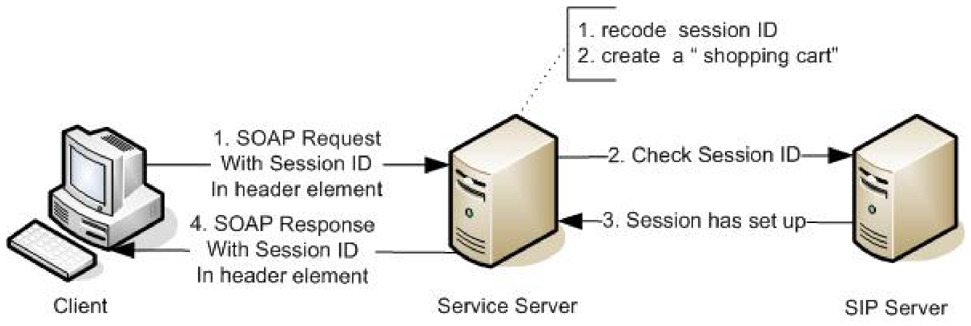


Figure 16 Web Services Invoking between the Client and the Service Server

4.2.7 Session Terminating

Once a Web service user finishes a video shopping activity, he or she should terminate the current active session or exit the client application system. During this procedure, the SIP client first sends a BYE message to the SIP server. After the SIP server receives this message, it informs the service server that the user's session is terminated and then sends a 200 OK message to the SIP client. As a result, the session established between the client application and SIP server is terminated and the session ID record is deleted from both the SIP server and the Web service. The process of terminating a session is shown in Figure 17.

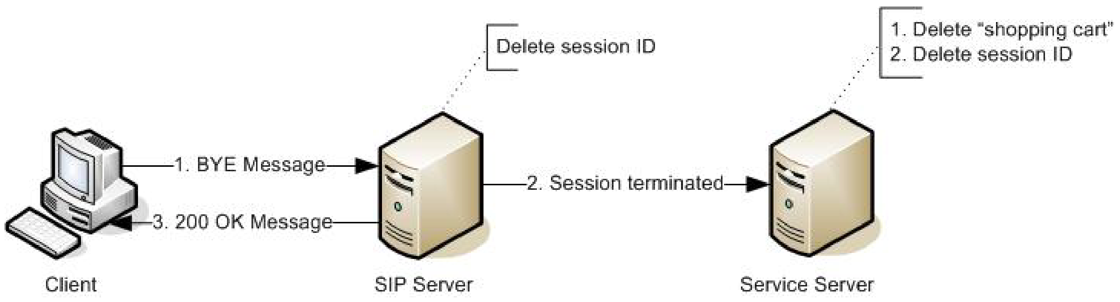


Figure 17 Session Termination between Client and Online Video Shopping Web Service System

4.3 System Design

4.3.1 Package Diagrams of the Online Video Shopping Web Service System

The packages of the system consist of three layers. The first layer is the *sipsoapsession* package. This package includes four sub-packages: a *sip* package, an *application* package, a *soapheader* package and an *util* package, which comprise the second layer of the package. Figure 18 shows the first two package layers of the

Online Video Shopping Web Service System.

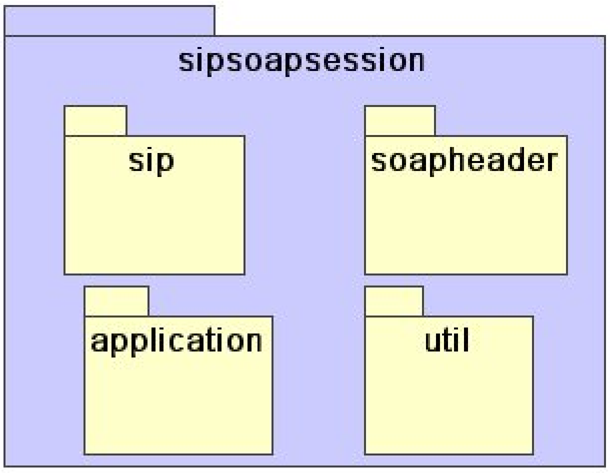


Figure 18 The First two package layers of the Online Video Shopping Web Service System

There of the second layer packages (*sip* package, *soapheader* package and *application* package) contain third layer packages (See Figure 19, 20, 21).

- ***Sip package***: This package includes three third layer packages. They are a *client* package, a *message* package and a *server* package. Figure 19 illustrates the package diagram of the *sip* package. The *client* package includes the class for implementing SIP client which can be used for sending and receiving SIP messages between the Web service user and SIP server, such as session establishment messages and session termination messages. The *message* package contains the classes for generating and processing SIP messages which are transmitted between the SIP client and the SIP server. The SIP server implementation classes are included in the *server* package.

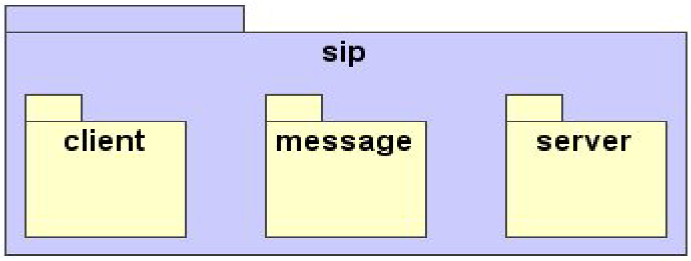


Figure 19 SIP Package Diagram

- ***Application package***: There are three sub-packages contained in this package and they are a *client* package, a *server* package and a *commonobj* package. As demonstrated in Figure 20, the classes included in the *commonobj* package are used by both the client and the server packages. The classes in the *client* package are composed the system client application. All the Web services implementation is enclosed in the *server* package.

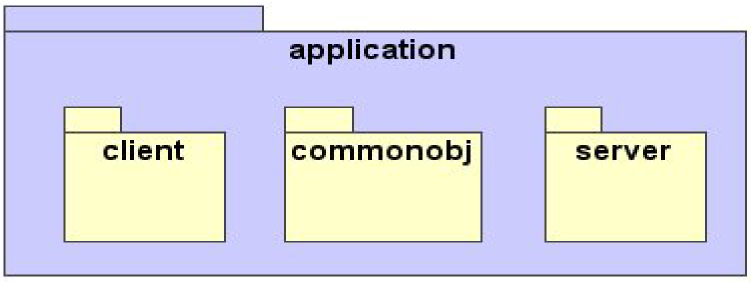


Figure 20 Application Package Diagram

- ***Soapheader package***: *axisheader* package is the only sub-package contained in the *soapheader* package and it is used to generate an AXIS soap header element that can contain the session ID within it (See Figure 21). This package can be extended by adding more sub-packages which can generate soap header element for different SOAP servers.

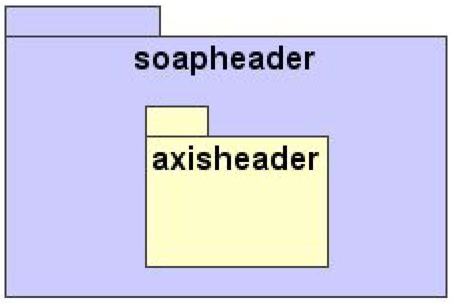


Figure 21 Soapheader Package Diagram

- ***Util package***: This package contains a session hash table that can be used for recording and managing the session information by SIP server and Web services.

4.3.2 Class Diagrams for Online Video Shopping Web Service System

This project focuses on the design and implementation of a SIP server and a video store Web service. This section first illustrates the SIP server design, and following that the design of the video store Web services will be introduced.

❖ SIP Server Class Diagram

As shown in Figure 22, ten classes coming from different packages are involved in the design of the SIP server. *SessionTable* belongs to the *sipsoapsession.util* package. *MessageProcess*, *MessageGenerator* and *MessageProcessResult* classes come from the *sipsoapsession.sip.message* package. *SIPServer*, *SIPMsgSockHandler*, *WebMsgServer*, *WebMsgSockHandler*, and *ServiceClient* are in the *sipsoapsession.sip.server* package. The *Thread* class comes from J2SE API. The description of each class is detailed below:

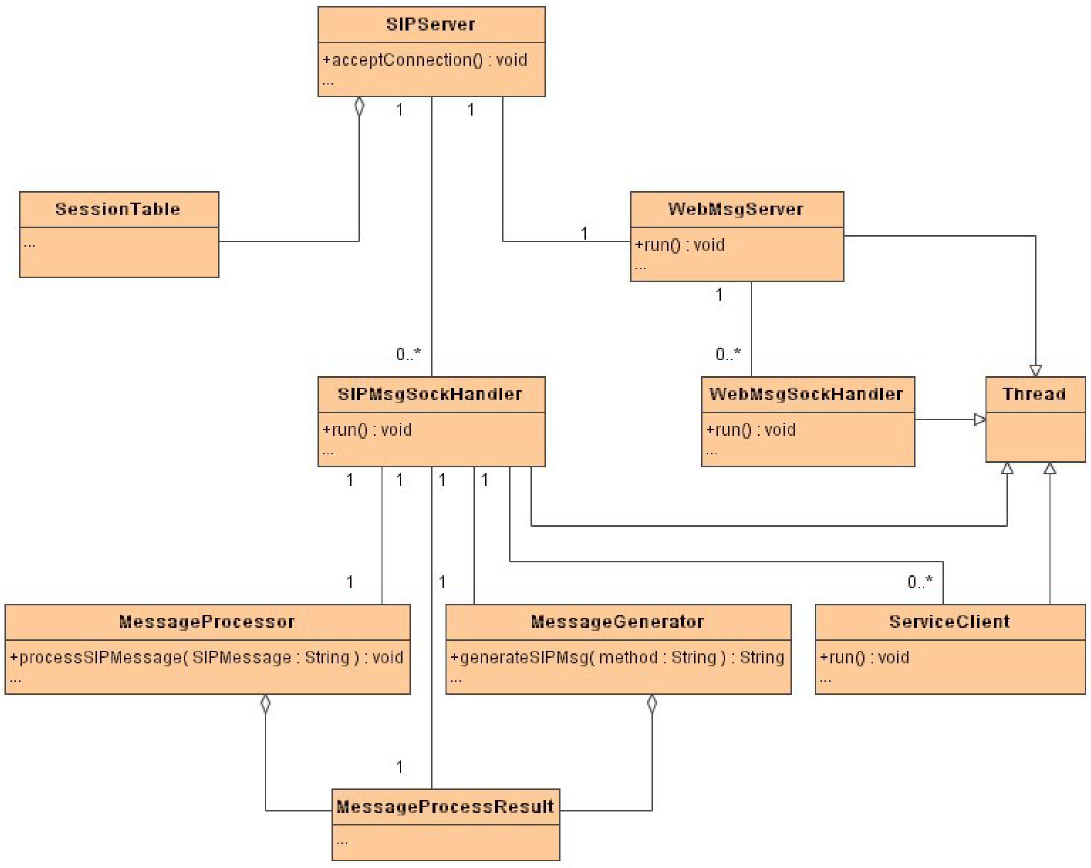


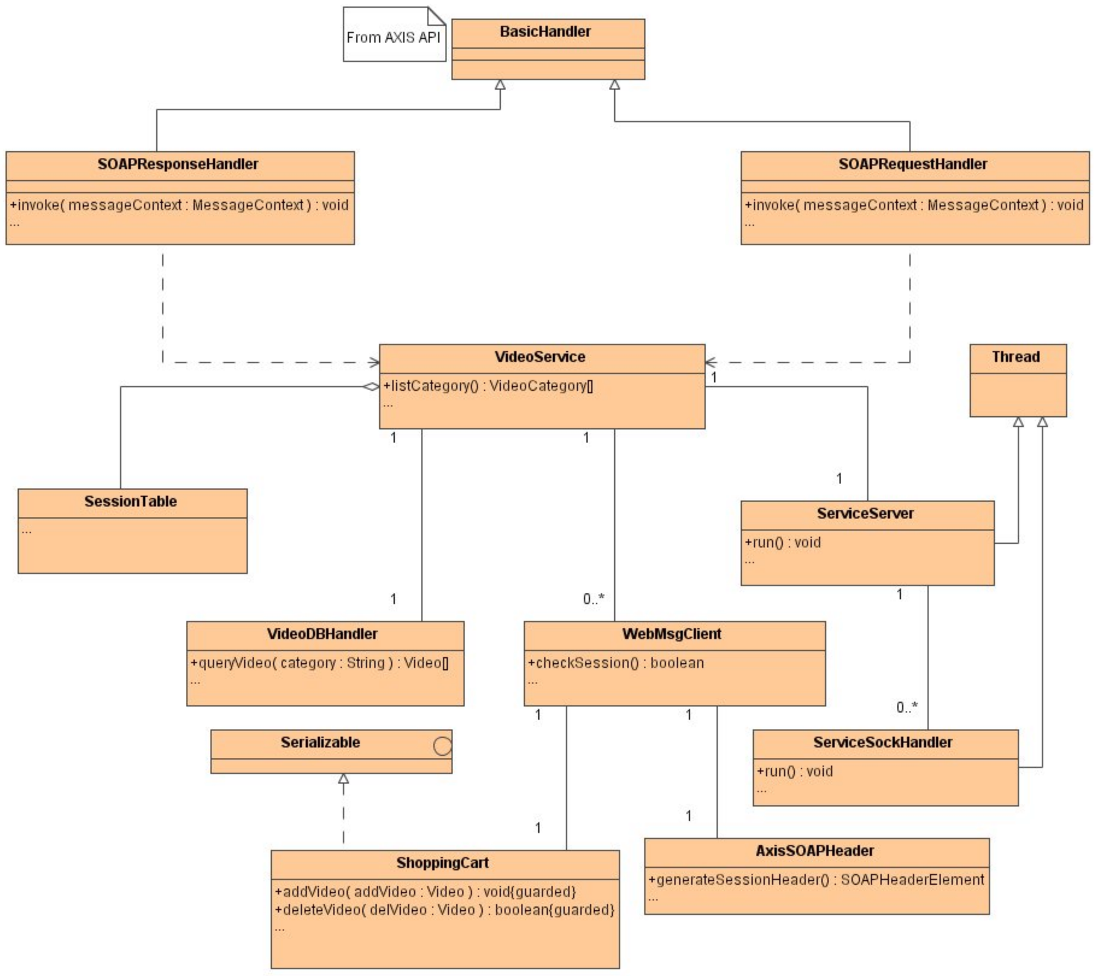
Figure 22 SIP Server Part Class Diagram

- *SessionTable*: this class acts as a record table for the SIP server to store the active session information, such as session ID, Web service IP address and port. Once a session is set up between the SIP client and the SIP server, the SIP server will record the session ID in its *SessionTable* for later use.
- *MessageProcess*: this class is responsible for processing the SIP messages sent between the SIP client and the SIP server. Some of the processing results will be saved in *MessageProcessResult* for later use.
- *MessageProcessResult*: this class can save the processing results of the *MessageProcess* class. The results will be used by the *MessageGenerator* class for generating SIP response messages or final SIP messages.

- *MessageGenerator*: this class is responsible for generating SIP messages. Sometimes it needs the information saved in *MessageProcessResult* to generate SIP messages.
- *SIPServer*: this class is used to accept messages sent from SIP clients. When it receives a new SIP message, *SIPserver* will start *SIPMsgSockHandler* to deal with the messages. In addition, *SIPServer* creates a *WebMsgServer* when it starts.
- *SIPMsgSockHandler*: this class is a subclass of the *Thread* class, and is started by *SIPServer* for handling SIP messages sent from a SIP client. It can make use of *MessageProcess*, *MessageGenerator* and *MessageProcessResult* classes to process and generate SIP messages. Additionally, when a session is terminated, it will start a *ServiceClient* class to inform a corresponding Web service.
- *WebMsgServer*: this class is a subclass of the *Thread* class, and is started by *SIPServer*. It is responsible for accepting a connection from the Web services and starting a *WebMsgSockHandler* to handle the messages from Web services.
- *WebMsgSockHandler*: this class is a subclass of the *Thread* class, and is started by *WebMsgServer*. Its functions are handling checking session requests from Web services and updating the SIP server *SessionTable* to record the Web services IP addresses and ports for establishing sessions.
- *ServiceClient*: this class is a subclass of the *Thread* class, and is started by *SIPMsgSockHandler*. By reading the information saved in the SIP server *SessionTable*, *ServiceClient* can inform a corresponding Web service that a session with the Web service is terminated.

❖ Video Store Web Service Class Diagram

Figure 23 is the class diagram of the Video Store Web Service. The Video Service consists of twelve classes and one interface. The *BasicHandler* class comes from AXIS API. The *Thread* class and *Serializable* interface are from J2SE API. As introduced above, the *SessionTable* class belongs to *sipsoapsession.util* package. *AxisSOAPHeader* class is in the *sipsoapsession.soapheader.axisheader* package. The remaining classes are owned by the *sipsoapsession.application.server* package. The detail description of important classes is listed below:



From AXIS API
BasicHandler
SOAPResponseHandler
+invoke(messageContext : MessageContext) : void
SOAPRequestHandler
+invoke(messageContext : MessageContext) : void
VideoService
+listCategory() : VideoCategory[]
Thread
SessionTable
ServiceServer
+run() : void
VideoDBHandler
+queryVideo(category : String) : Video[]
WebMsgClient
+checkSession() : boolean
Serializable
ServiceSockHandler
+run() : void
ShoppingCart
+addVideo(addVideo : Video) : void{guarded}
+deleteVideo(delVideo : Video) : boolean{guarded}
AxisSOAPHeader
+generateSessionHeader() : SOAPHeaderElement
1
0..*

Figure 23 Video Service Class Diagram

- *SOAPRequestHandler*: this class is the subclass of the *BasicHandler* class. It is used to inform *VideoService* of the session ID contained in the SOAP request messages. It also outputs the SOAP request messages into a log file called VideoService.log.
- *SOAPResponseHandler*: this class is the subclass of the *BasicHandler* class. It is able to insert session ID into SOAP response messages

generated by AXIS, and output the response messages into the VideoService.log file.

- *VideoService*: this class is the video store Web service implementation. For every web user's request, it can make use of *SessionTable* to save the session information, utilize *VideoDBHandler* to handle the request, and generate the corresponding result to the client. In addition, it creates a *ServiceServer* object to accept the messages from SIP server, and starts a *WebMsgClient* object to check a new session ID with the SIP server.
- *VideoDBHandler*: this class can be used to access the data saved in the video store database according to client requests.
- *ServiceServer*: this class is a subclass of the *Thread* class. It is started by *VideoService*. For every connecting message sent from SIP server, *ServiceServer* will start a *ServiceSockHandler* object to handle it.
- *ServiceSockHandler*: this class is a subclass of the *Thread* class. It can handle session termination messages sent from the SIP server.
- *WebMsgClient*: this class is a subclass of the *Thread* class. It is started by *VideoService* to check a new session ID with the SIP server. If the session has been set up, *WebMsgClient* will add an *AxisSOAPHeader* and a *ShoppingCart* object in the *VideoService SessionTable*.
- *ShoppingCart*: this class can be used to store the video items selected by the Web service users.
- *AxisSOAPHeader*: this class can generate a SOAP header element that contains the session ID within it. Since AXIS was used as the SOAP server, the generated SOAP header element is defined by AXIS API.
- *SessionTable*: similar to the function in SIP server design, this class can be used to store the session information for *VideoService*. This information includes session ID, *AxisSOAPHeader* and *ShoppingCart*.

4.3.3 Sequence Diagrams of the Online Video Shopping Web Service System

In this section, three sequence diagrams demonstrate the interactions between the client GUI application, the SIP server and the video Web service in the system in the following scenarios:

- ♦ Establishing a new session
- ♦ Invoking a Web service
- ♦ Terminating a session

Establishing a new session

Establishing a new session between a Web service user and the SIP server consists of following steps (see Figure 24):

1. *GuiIntial* on the client side uses the *SIPClient* to setup a connection with *SIPServer*.
2. *SIPServer* accepts the connection.
3. *SIPServer* starts a *SIPMsgSockHandler* to handle the messages sent from *SIPClient*.
4. *SIPClient* sends an INVITE message to *SIPMsgSockHandler*.
5. *SIPMsgSockHandler* responds with a 200 OK message to *SIPClient*.
6. *SIPClient* generates an ACK message to *SIPMsgSockHandler*, and then a new session between *GuiIntial* and the SIP server is established.

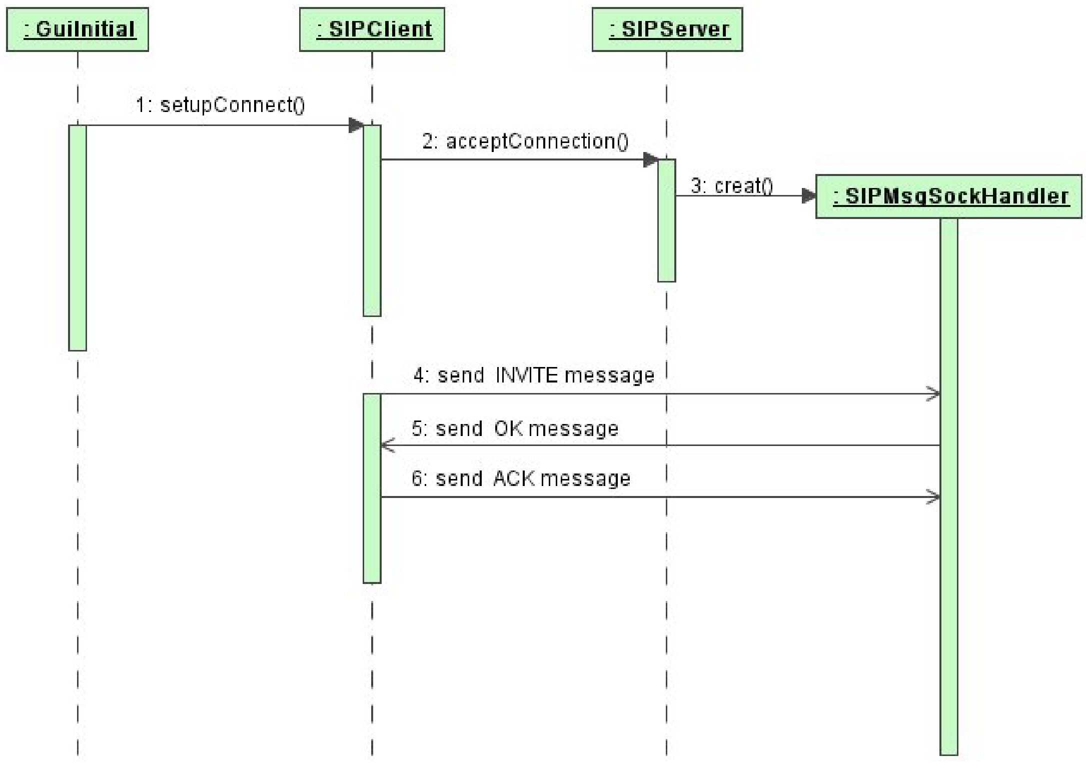


Figure 24 Session Establishing Sequence Diagram

Invoking a Web service

Figure 25 takes an invoking "list video category" function as an example to illustrate the procedure of invoking a Web service.

1. Once the *GuiIntial* sets up a session with the SIP server, the Web service user will get a session ID. Then *GuiIntial* tells *VideoClient* that it wants to see the video category by invoking *VideoClient's listcategory()* method.
2. *VideoClient* generates the SOAP request, which contains the session ID in its header element and sends the request to the AXIS Server.
3. *SOAPRequestHandler*, which is inserted into the SOAP request chain in AXIS, handles the request by calling the *VideoServer's setSOAPHandler()* method to check whether the session ID is set up with the SIP server.
4. *VideoService* creates a *WebMsgClient* object to check the session ID with the SIP server.

5. *WebMsgClient* connects with the *WebMsgServer*, which is started by the SIP server, to accept the checking session connection from the Web service.
6. The *WebMsgServer* then starts a *WebMsgSockHandler* object to handle the messages sent from *WebMsgClient*.
7. *WebMsgClient* sends the session ID to *WebMsgSockHandler*.
8. *WebMsgSockHandler* checks the session ID with the SIP session table and then sends the "session set up" result back to *WebMsgClient*.
9. After *WebMsgClient* receives the checking result, it will send the *VideoService* IP address and a port number to the SIP server. This information will be used to inform *VidoService* when the session is terminated.
10. When the *WebMsgSockHandler* receives the IP address and the port number, it will insert them into the SIP session table for later use.
11. WebMsgClient also needs to update the *VideoService* session table by adding the session ID, a new shopping cart and a corresponding SOAP header. The shopping cart created here will be used to keep the video items for the Web service users.
12. After that, *WebMsgClient* stops the connection with *WebMsgSockHandler*.
13. *WebMsgClient* returns the checking session result to *VideoService* with a boolean value.
14. Then the boolean value will be sent from *VideoService* to *SOAPRequestHandler*.
15. After receiving a positive result, the *listategory()* method call will be delivered to *VideoService*.
16. *VideoService* makes use of *VideoDBHandler* to query the database of the video store and returns the result.
17. During the procedure of generating a SOAP response to the client, a *SOAPResponseHandler* is used to get the session ID from the *VideoService* through invoking the *setSOAPHeader* method of *VideoService*.
18. And then the *SOAPResponseHandler* will add the session ID into the SOAP header element of the SOAP response message.

19. The SOAP response message is received by *VideoClient*.

20. Finally, the Web service user can see the video category list.

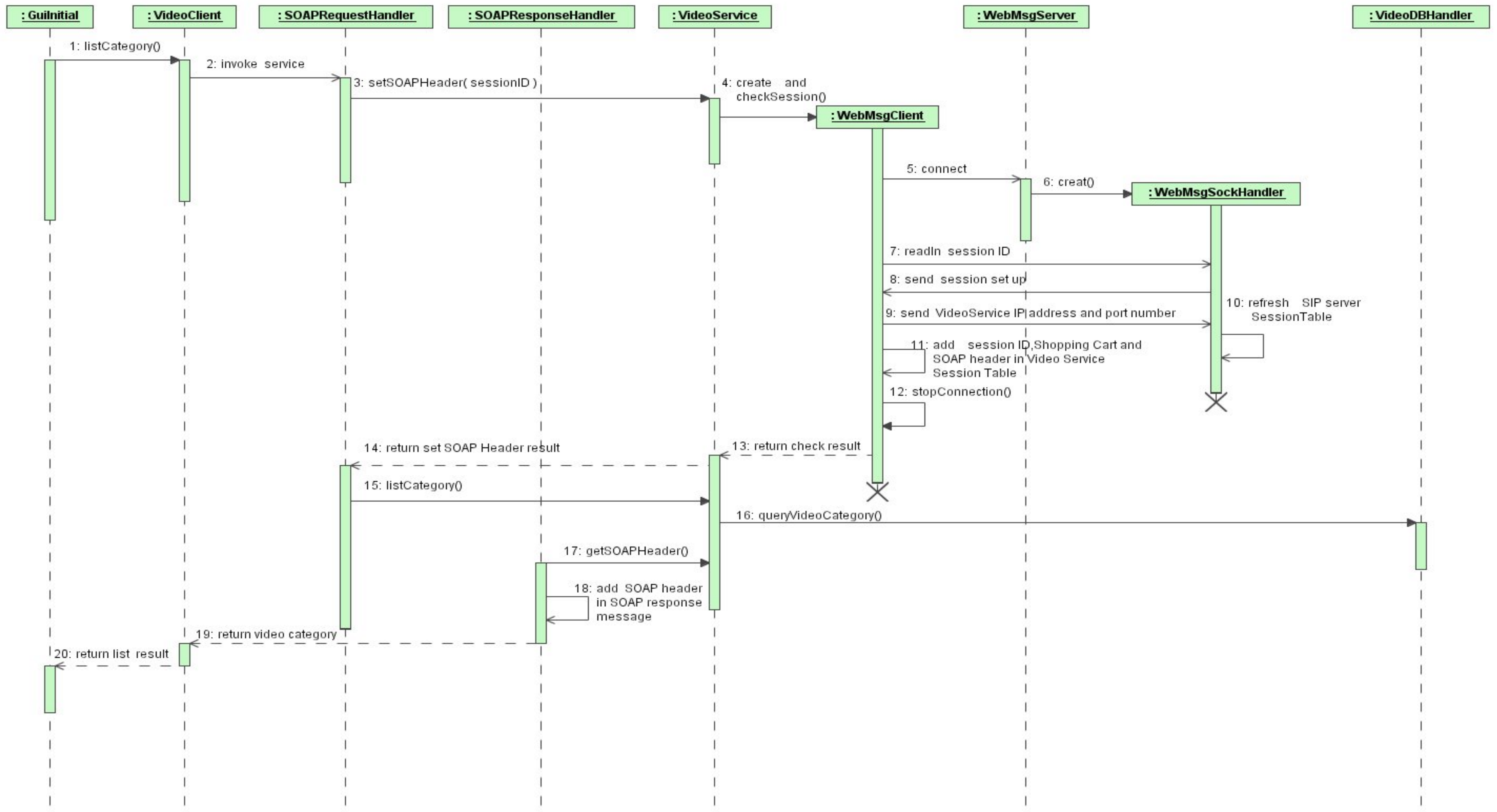


Figure 25 Web Service Invoking Sequence Diagram

To sum up, there are five stages in the procedure of invoking a Web service:

- ➢ The first stage (steps1-2): the Web service user sends a SOAP request message to a Web service.
- ➢ The second stage (steps 3-14): the invoking Web service checks the session ID with the SIP server.
- ➢ The third stage (steps15-16): the service method is invoked and the result is generated for the Web service user.
- ➢ The fourth stage (steps 17-18): the session ID is added into the SOAP response message.
- ➢ The fifth stage (steps 19-20): the final result is shown to the Web service user.

Terminating a session

When a Web service user accomplishes his or her video shopping activities, he or she needs to terminate the session. Figure 26 shows the steps involved in terminating a session.

1. *GuiIntial* informs the *SIPClient* that it wants to terminate the current session.
2. *SIPClient* then sends a BYE message to the SIP server, and the message is handled by the *SIPSockHandler*, which is created in the session establishing procedure.
3. *SIPSockHandler* creates *ServiceClient* to inform *VideoService* that a session is terminated.
4. The *ServiceClient* connects with *ServiceServer*, which is created by *VideoService*, to receive session terminating messages from the SIP server.
5. The *ServiceServer* starts a *ServiceSockHandler* to deal with the messages from *ServiceClient*.
6. The *ServiceSockHandler* reads the terminating session ID from *ServiceClient*.
7. *ServiceSockHandler* then deletes the terminating session ID from the session table of *VideoService*. As a result, the shopping cart and the SOAP header information kept in the session table will also be deleted.

8. The *SIPMsgSockHandler* on the *SIPServer* side removes the terminating session ID from the *SIPServer* session table.
9. After that, the *SIPMsgSockHandler* sends a 200 OK message to *SIPClient*, which means that the session has been terminated.
10. Finally, *SIPClient* stops the connection with *SIPMsgSockHandler*.

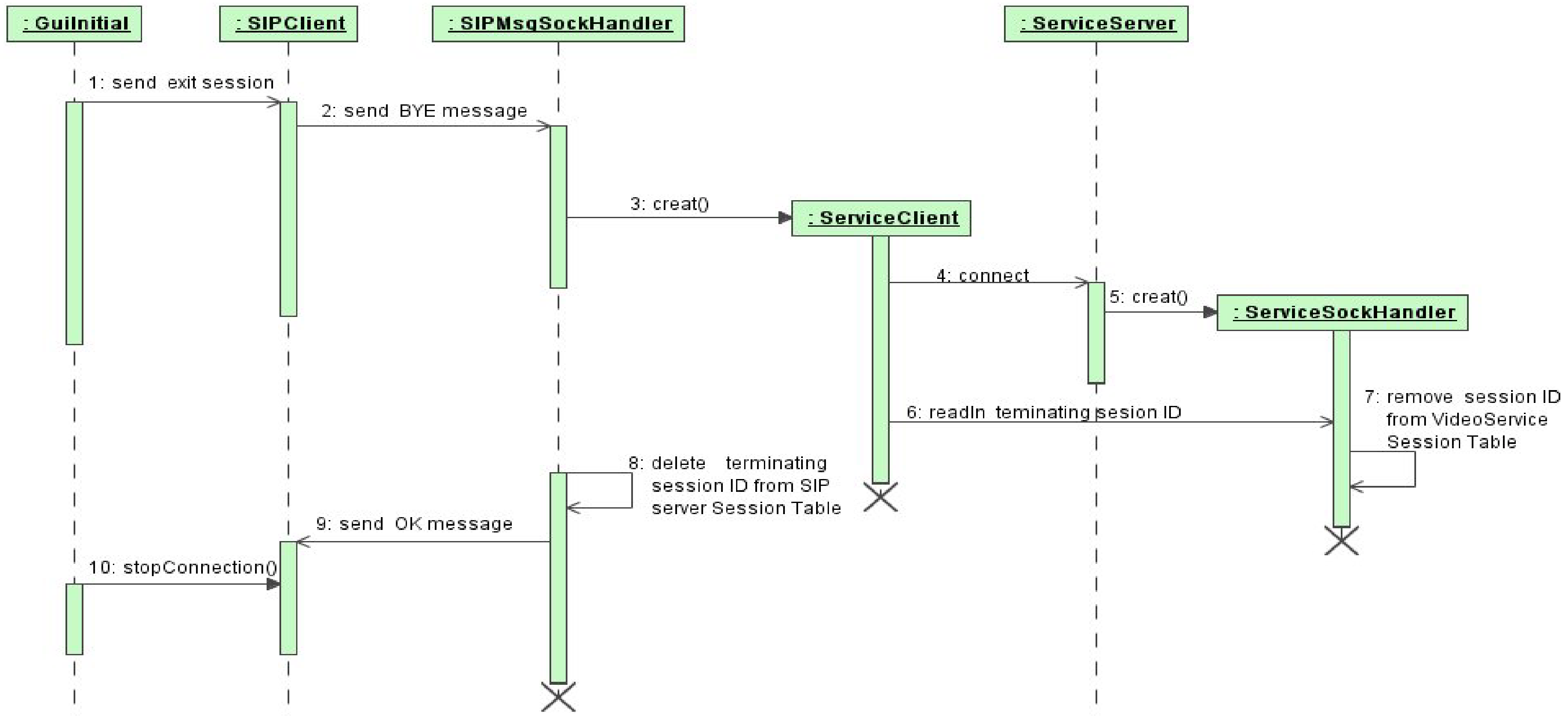


Figure 26 Session Terminating Sequence Diagram

4.4 Deployment

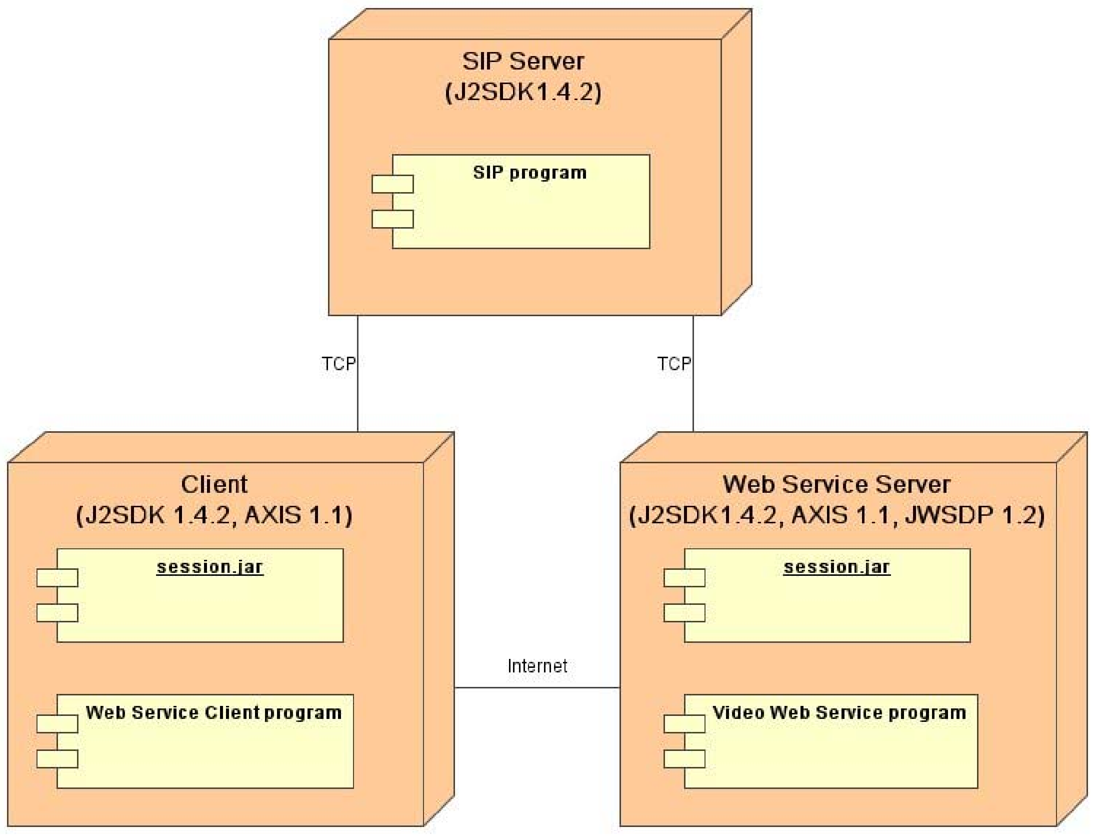


Figure 27 Deployment Diagram of the Online Video Shopping Web Services System

The deployment diagram of the Online Video Shopping Web Services System is shown in Figure 27. The session jar file needs to be put into an accessible directory on both the client and Web service server sides. The session jar file needs to be copied to common\lib directory, which is under the JWSDP1.2 home directory, in order to let the HTTP server (Tomcat contained in JWSDP 1.2) locate it. The Video Web Service program includes *sipsoapsession.application.server* and *sipsoapsession.application.commonobj* packages. The packages that the Web Service Client program needs to have are *sipsoapsession.application.client* and *sipsoapsession.application.commonobj*. The PC which is running the SIP server should contain *sipsoapsession.sip* package. The packages included in *session.jar* are the *sip* package, the *soapheader* package and the *util* package. All the computers in this system should support Java 1.4.2 and network connection.

4.5 Implementation Discussion

4.5.1 SIP Server Implementation

❖ SIP Transport Protocol

From the SIP Protocol Stack introduced in chapter 2, it can be seen that SIP can utilize TCP and UDP to transfer its messages through the Internet. Using UDP to carry SIP messages costs less set up connection time than using TCP, and reduces the Internet traffic jam possibility. However, the lack of handshaking or acknowledgement in UDP transport means that a datagram could be lost along with a SIP message. When this happens in an E-commerce application, it will cause a serious security problem. In addition, HTTP uses TCP as its transport protocol to provide a reliable connection service to Web users. Therefore, in our system, TCP is absorbed as the SIP transport protocol to supply a stable connection for both the Web service users and providers.

❖ Standard Session ID

The session ID is the key point for accomplishing session management in this system. Hence, generating a cyber unique identification when the session is established is the most important element for SIP implementation. In order to express the session ID in a standard manner, we use the “o” field of SDP and ISO 8601 Date/Time Representations [53, 54] are used to generate the session ID. According to SDP specification, the format of the session ID is:

YYYYMMDDhhmmss@<SIP client IP address>

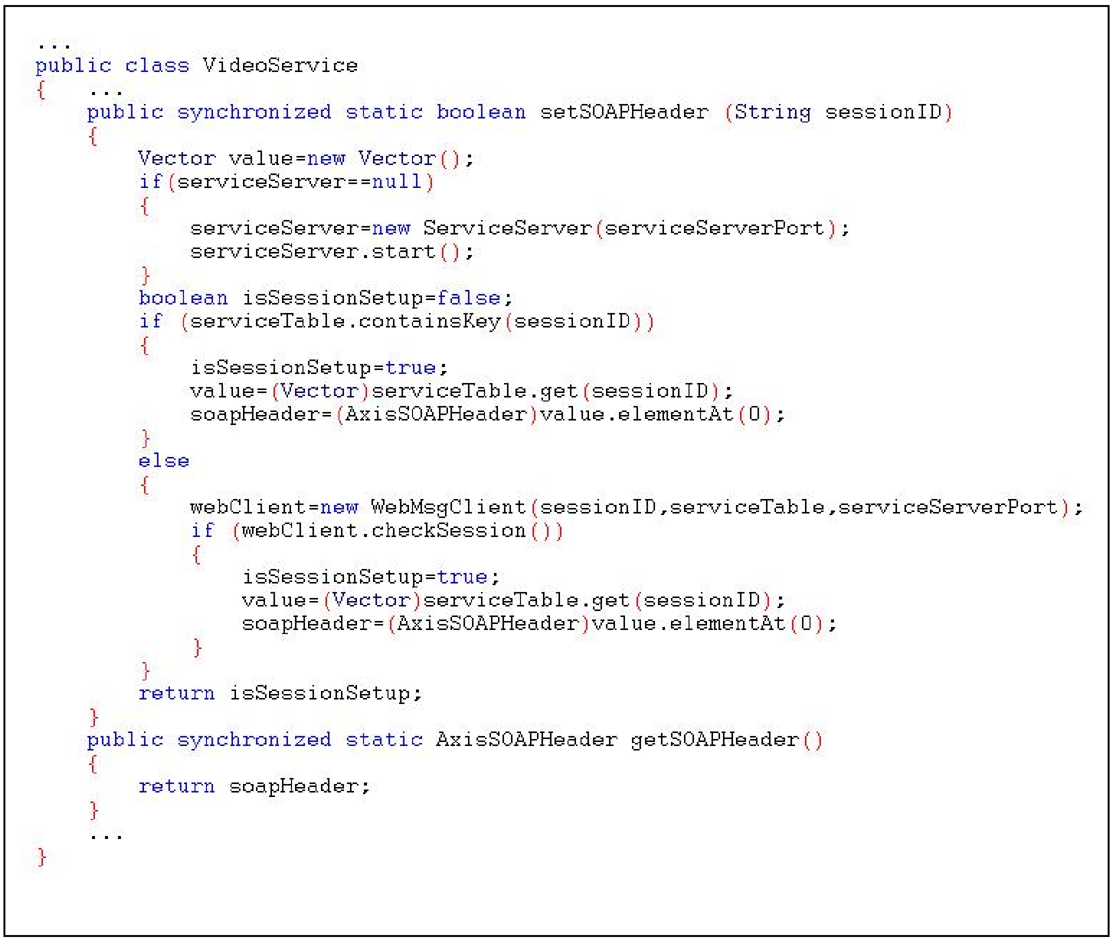
For example, a SIP client whose IP address is 192.168.0.100 sets up a session with a SIP server at 5:06:30 am, 19 December, 2004. The session ID is:

20041219050630@192.168.0.100.

4.5.2 The Communication Implementation of SOAP Requests and Web Services

Normally, after a SOAP server receives a SOAP request, it will analyze the SOAP message and then invoke a corresponding back-end Web service program. In other words, the Web service program cannot directly access the SOAP request. But in this system, the session ID enclosed in SOAP message header element needs to be managed by the Web service. To accomplish the communication between SOAP messages and the back-end Web Service by using AXIS SOAP server, four steps need to be done:

- Writing static methods in the *VideoService* class to implement session ID checking and getting functions. (See Figure 28) The method *setSOAPHeader* is the session ID checking function and the method *getSOAPHeader* is the session ID getting function.



```
...
public class VideoService
{   ...
    public synchronized static boolean setSOAPHeader (String sessionID)
    {
        Vector value=new Vector();
        if(serviceServer==null)
        {
            serviceServer=new ServiceServer(serviceServerPort);
            serviceServer.start();
        }
        boolean isSessionSetup=false;
        if (serviceTable.containsKey(sessionID))
        {
            isSessionSetup=true;
            value=(Vector)serviceTable.get(sessionID);
            soapHeader=(AxisSOAPHeader)value.elementAt(0);
        }
        else
        {
            webClient=new WebMsgClient(sessionID,serviceTable,serviceServerPort);
            if (webClient.checkSession())
            {
                isSessionSetup=true;
                value=(Vector)serviceTable.get(sessionID);
                soapHeader=(AxisSOAPHeader)value.elementAt(0);
            }
        }
        return isSessionSetup;
    }
    public synchronized static AxisSOAPHeader getSOAPHeader()
    {
        return soapHeader;
    }
    ...
}
```

Figure 28 Session ID Checking and Getting Code Segment in VideoService

- Implementing two subclasses of AXIS *BasicHandler*. One class is used to inform *VideoService* of the session ID in the SOAP request. The other class adds session ID in the SOAP response (See Figure 29 and 30). The class *SOAPRequestHandler* in Figure 29 uses the *setSOAPHeader* method in *VideoService* to inform the back-end Web service to check the session ID in the SOAP request. The function of class *SOAPResponseHandler* in Figure 30 is adding session ID in SOAP response messages by using the *getSOAPHeader* method in *VideoService*.

```
...
public class SOAPRequestHandler extends BasicHandler{
    ...
   public void invoke (MessageContext messageContext) throws AxisFault{
      try{
            Message reqMessage=messageContext.getRequestMessage();
            SOAPEnvelope env=reqMessage.getSOAPEnvelope();
            Element envElement=env.getAsDOM();
            //deal with header
            SOAPHeaderElement SOAPhead = env.getHeaderByName
                        ( http://www.sipsession.com/sipsession , "SIPSession" );
            Iterator iterator=SOAPhead.getChildElements();
            String sessionID=null;
            boolean setResult=false;
            while(iterator.hasNext()){
            SOAPElement headElement=(SOAPElement)iterator.next();
            if (headElement.getElementName().getLocalName().equals("sessionID"))
            {
                sessionID=headElement.getValue();
                setResult=VideoService.setSOAPHeader(sessionID);
                if (!setResult){
                    throw new AxisFault
                    ("Request Handler: The client has not set up connection with SIP server");
                }
            }
            ...
        }catch(Exception e){
            throw AxisFault.makeFault(e);
        }
    }
}
```

Figure 29 SOAPRequestHandler Class Code Segment

```
...
public class SOAPResponseHandler extends BasicHandler{
    ...
     public void invoke(MessageContext messageContext)throws AxisFault{
    try{
            long currentTimeMillis=System.currentTimeMillis();
            Long startTime=(Long)messageContext.getProperty("StartTime");
            long startTimeMills=startTime.longValue();
            float duration=((float)(currentTimeMillis-startTimeMills))/1000;
            String strDuration="Time to make call : "+String.valueOf(duration)+" seconds";
            //add header
            AxisSOAPHeader header=VideoService.getSOAPHeader();
            Message resMessage=messageContext.getResponseMessage();
            SOAPEnvelope env=resMessage.getSOAPEnvelope();
            env.addHeader(header.generateSessionHeader());
            ...
        }catch(Exception e){
            throw AxisFault.makeFault(e);
        }
    }
}
```

Figure 30 SOAPResponseHandler Class Code Segment

- Inserting *SOAPResponseHandler* and *SOAPResponseHandler* into AXIS request and response handlers by specifying them in a *deploy.wsdd* file. (Shown in Figure 31) The WSDD (Web Service Deployment Descriptor) file is an XML-based format and is used to configure the properties of the AXIS engine [3]. The <handler> tag defines the handlers that need to be specified for the AXIS engine. The <requestFlow> and <responseFlow> tell the AXIS engine where the handler should be put.

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
            xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <handler name="SOAPRequestHandler"
          type="java:sipsoapsession.application.server.SOAPRequestHandler">
     <parameter name="logfilename"
                value="d:\\java\\project\\sipsoapsession\\application\\VideoService.log"/>
  </handler>

  <handler name="SOAPResponseHandler"
          type="java:sipsoapsession.application.server.SOAPResponseHandler">
     <parameter name="logfilename"
                value="d:\\java\\project\\sipsoapsession\\application\\VideoService.log"/>
  </handler>

  <service name="VideoService" provider="java:RPC">
      <requestFlow>
          <handler type="SOAPRequestHandler"/>
      </requestFlow>
      <responseFlow>
          <handler type="SOAPResponseHandler"/>
      </responseFlow>
      <parameter name="className" value="sipsoapsession.application.server.VideoService"/>
      <parameter name="allowedMethods" value="listCategory listVideo addToShoppingCart
                                        delFromShoppingCart getVideoItems purchase"/>

       ...
  </service>
</deployment>
```

Figure 31 Specifying Handlers in Deploy.wsdd File

- Deploying the Web service. The following command can be used to deploy the Video Web service in a DOS window.

```
java org.apache.axis.client.AdminClient -l
http://localhost:8080/axis/services/AdminService deploy.wsdd
```

After these four steps, the *SOAPResponseHandler* and *SOAPResponseHandler* have been deployed into AXIS request and response handler flows, and become the bridge for SOAP messages and back-end Web Service communication.

4.6 Summary

This chapter has presented the architecture of using SIP to manage sessions for Web services and the technologies used in an Online Video Shopping Web Service System. We use SIP technology to set up and terminate sessions for Web services. The set up session ID will be kept in the SOAP messages sent between the users and Web services. Therefore, the Web services can distinguish different users from the session ID. A session table is applied to manage the session on Web services. It keeps the session ID and shopping cart as a key – value pair, as a result, the Web services can handle all users' requests by administrating the session table.

In order to deliver the session ID from the SOAP message header to the back-end Web services, AXIS as the SOAP server provides an extensible message processing system for Web service programmers. By extending the basic handler class and then deploying to the AXIS engine, programmers can easily add custom functionalities. If applying this system architecture to other SOAP servers, they would have to have similar interface or function for programmers to accomplish the session information exchange. The next chapter presents the E-banking Web Services System which is based on the session management mechanism of this system to achieve transaction management for Web services.

Chapter 5 Transaction Management on Web services by Using SIP

5.1 Introduction

In the previous chapter, the mechanism of using SIP to manage the session on Web services has been presented. The Online Video Shopping Web Services System has been implemented. This chapter will present the transaction management mechanism on Web services which is based on the previous implementation described in chapter 4. An E-banking Web Services System will be used to illustrate the implementation of the mechanism.

This chapter will first provide the system analysis of the E-banking Web service system in section 5.2, followed by the system design in section 5.3. After this, the system deployment will be illustrated in section 5.4. The system testing is presented in section 5.5. A summary will be given at the end.

5.2 System Analysis

5.2.1 Introduction of the E-banking Web service system

The goal of this stage is to accomplish transaction management on Web services. Thus, the E-banking Web Services System focuses on the implementation of money transfer transaction management between two banks. Bank A and Bank B Web services will be implemented which provide money transferal functions. The system client can choose an account from one of these banks to transfer money to an account in another bank.

The previous stage of this research has provided a session management mechanism for Web services, which is the first step to complete transaction management. Therefore, the mechanism will be used in the implementation of the transaction management in the E-banking Web services system. However, the session ID used in the previous stage cannot be used to identify a transaction between the two banks because a client might create many transactions within a session. In order to accomplish transaction management for Web services, a transaction manager needs to be brought in which is responsible for coordinating all the participants' activities in a transaction. In the following part of this section, the architecture of the system will be presented.

5.2.2 Architecture

Figure 32 shows the use case diagram of the system.

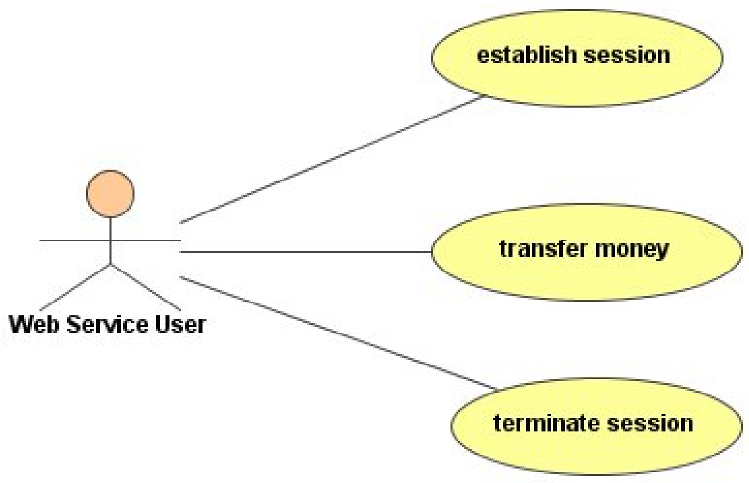


Figure 32 Use Case Diagram of Transaction Management on Web Services

In this system, the Web service user can set up a session with a SIP server, transfer money between two banks (Bank A and Bank B in the system), and terminate the session with the system. The architecture of the system is illustrated in Figure 33.

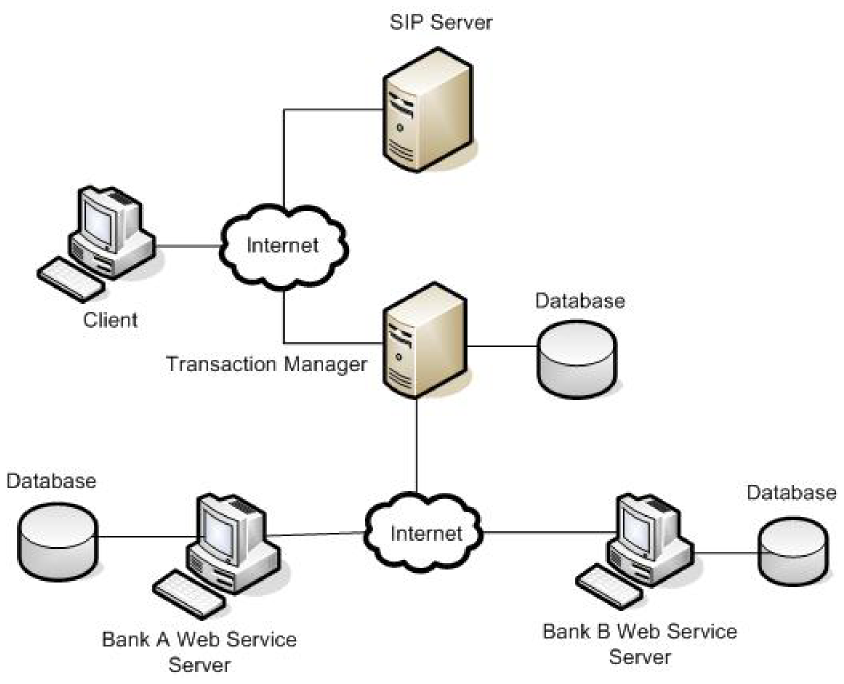


Figure 33 Architecture of the E-banking Web Services System

As with the previous system, the task of the SIP server is managing the session for all the Web services in the system. The Transaction Manager is a Web service that looks after all the transactions in the system. Bank A Web service server and Bank B Web service server are service servers that provide money transferal functions. The communication between Transaction Manager and SIP server and between Transaction Manger and the two bank Web service servers is through the Internet.

5.3 System Design

5.3.1 Package Diagram of the E-banking Web Service System

As shown in Figure 34, there are eight sub-packages included in the *sipsoaptransaction* package. The *banka* and the *bankb* packages contain classes for implementing Bank A and Bank B Web services respectively. The classes in the *client* package are for Web service users to invoke Web services and to transfer money between banks. The *transaction* package contains transaction manager implementation classes which can provide transaction coordination service for the participants. The *commonobj* package includes the *Account* class that will be used by both the two bank Web services and the transaction manager service. The classes in other packages, such as *sip* package, *soapheader* package and *util* package, are the same as the classes in the Online Video Shopping System. The E-banking Web service system still uses SIP to manage the session which is similar to the previous one. The difference is that this time SIP is used to manage the session for the transaction manager's Web service and not for the two bank Web services. The pivot of this system is the design and implementation of the transaction manager.

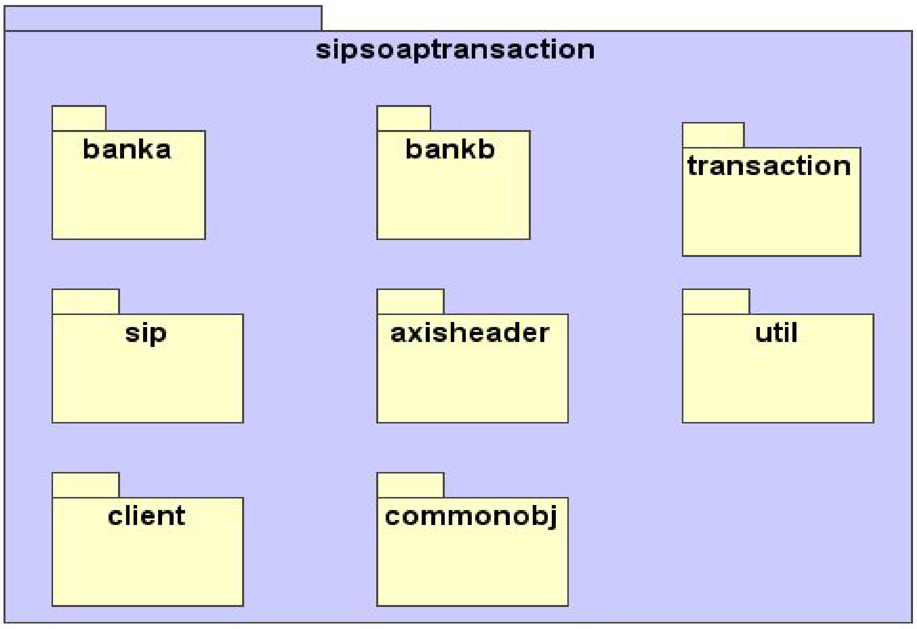


Figure 34 E-banking Web Services System Package Diagram

5.3.2 Class Diagram of Transaction Manager

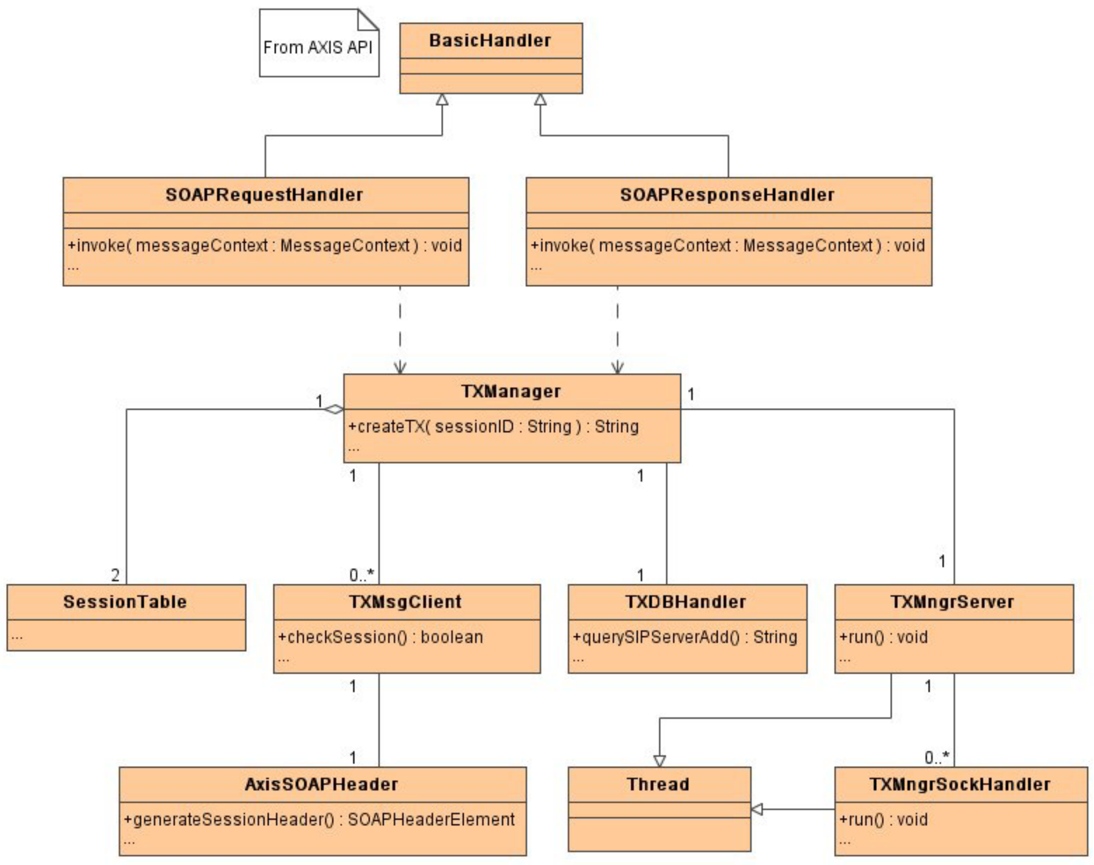

From AXIS API
BasicHandler
SOAPRequestHandler
+invoke(messageContext : MessageContext) : void
...
SOAPResponseHandler
+invoke(messageContext : MessageContext) : void
...
TXManager
+createTX(sessionID : String) : String
...
SessionTable
...
TXMsgClient
+checkSession() : boolean
...
TXDBHandler
+querySIPServerAdd() : String
...
TXMngrServer
+run() : void
...
AxisSOAPHeader
+generateSessionHeader() : SOAPHeaderElement
...
Thread
TXMngrSockHandler
+run() : void
...
1
2
0..*
1
1
0..*

Figure 35 Transaction Manger Class Diagram

Figure 35 shows the class diagram of the transaction manager. The composition of the transaction manager is very similar to the video service one in the previous project. It consists of eleven classes. The *BasicHandler* class comes from AXIS API, and the Thread class is from J2SE API. The *SessionTable* class belongs to package *sipsoaptransaction.util*, while *AxisSOAPHeader* class is in *sipsoaptransaction.soapheader.axisheader* package. The remaining classes belong to *sipsoaptransaction.transaction* package. In fact, the some of the class functions are the same as that of video store Web service. Table 5 lists the comparison of class functions between the video store Web service and the transaction manager.

Transaction Manager class name	Corresponding Video Store Web Service class name	Function Comparison between video store Web service and transaction manager
SOAPRequestHandler		Almost same. The only difference is that SOAPRequestHandler in transaction manager outputs SOAP request messages into a file named TXManager.log.
SOAPResponseHandler		Almost same. The only difference is that SOAPResponseHandler in transaction manager outputs SOAP response messages into a file named TXManager.log.
TXManager	VideoService	Different. TXManager implements all functions of a transaction manager. While VideoService implements all functions of a video store Web service.
TXDBHandler	VideoDBHandler	Same. They are both used to handle database query.
TXMngrServer	ServiceServer	Same.
TXMngrSockHandler	ServiceSockHandler	Same.
TXMsgClient	WebMsgClient	Almost Same. The only difference is that TXMsgClient put an initial transaction number in TXManager session table, but not a shopping cart.
AxisSOAPHeader		Same.
SessionTable		Almost same. Transaction manager owns two SessionTable objects. One of them is for managing session information, and the other is for managing transaction information.

Table 5 Classes Function Comparison between Video Store Web Service and Transaction Manager

5.3.3 Sequence Diagrams

This section focuses on introducing how the transaction manager coordinates all transaction participants. The sequence diagram in Figure 36 illustrates the process of transferring money successfully from Bank A to Bank B. It is assumed that the client *GuiInitial* has already set up a session with SIP server, and the transaction manager already has knows the session ID.

1. When *GuiInitial* gets the session ID from the SIP server, it invokes the *createTX()* method in *TXManager* (Transaction Manager) to create a new transaction.
2. Then *GuiInitial* gets a unique transaction ID from *TXManager*.
3. The *GuiInitial* can now invite transaction participants and it invites Bank A first to join the transaction.

4. Bank A then joins in the transaction by invoking *TXManager*'s *joinTX()* method.
5. *TXManager* returns the joining result to Bank A Web service with confirming information.
6. After Bank A Web service gets the confirmation result from *TXManager*, it gives *GuiInitial* an invitation response to indicate that it has joined in the transaction.
7. Then *GuiInitial* invites Bank B to join in the transaction and this is accomplished in steps 7, 8, 9, 10.
8. When *GuiInitial* gets all the invitation responses from Bank A and Bank B Web services, it tells *TXManager* to transfer money from a Bank A account to a Bank B account by invoking the *commit()* method in *TXManager*.
9. *TXManager* then gives the account information to Bank A and Bank B Web services and asks them to prepare for transferal of the money. After Bank A and Bank B Web services check the account information, they each give a "prepared" response to *TXManager* which declares that they have prepared for the money transfer. Steps 12, 13 and 14, 15 accomplish this procedure.
10. Once *TXManager* receives the "prepared" responses from both of the bank Web services, it commits the money transferal command to both of the bank Web services and then gets the transfer results from the Web services (See step 16, 17, 18, 19).
11. In steps 20 and 21, *TXManager* informs Bank A and Bank B Web services that the transaction has successfully finished.
12. Finally, *TXManager* sends the transferal money result to *GuiInitial*.

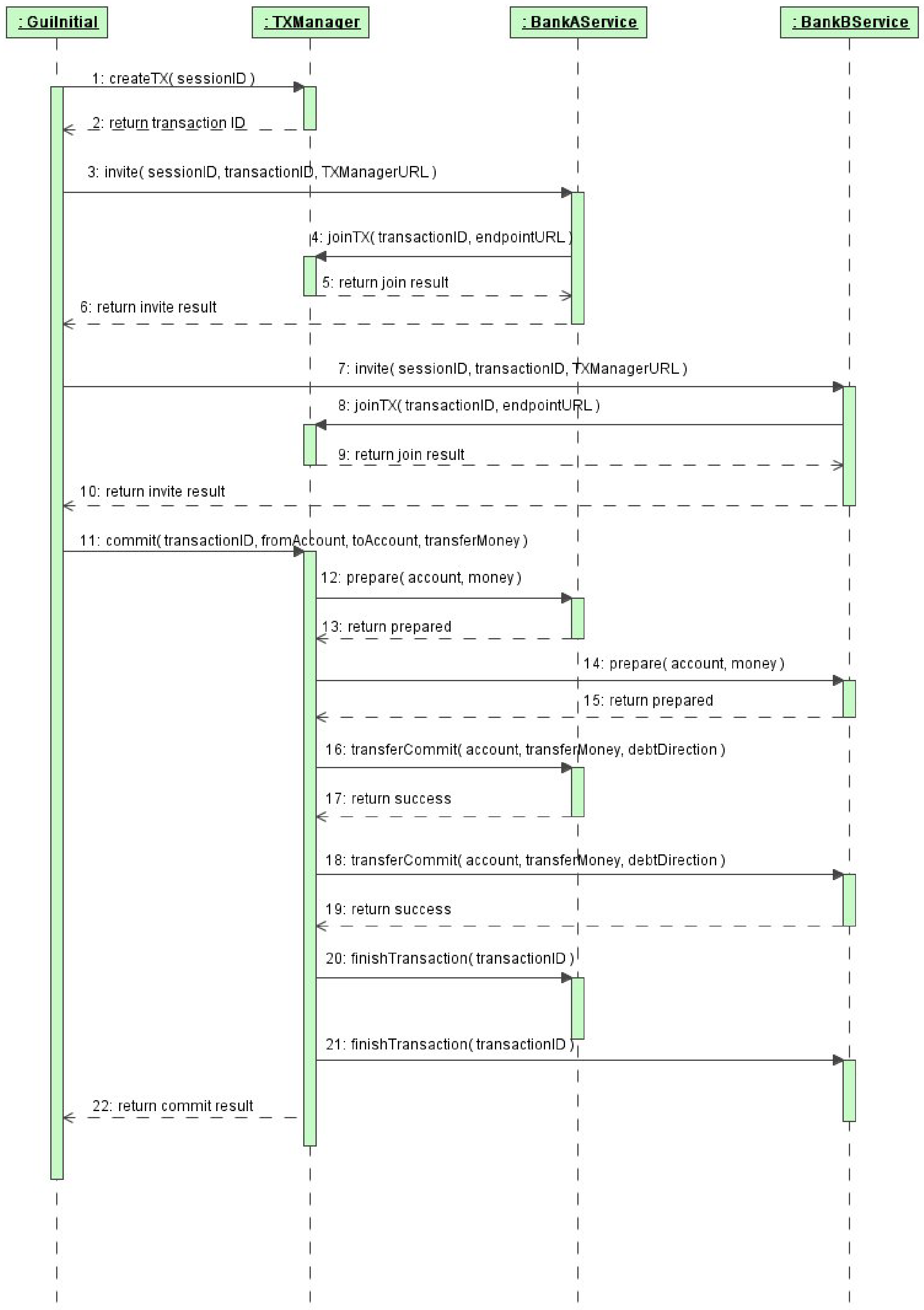
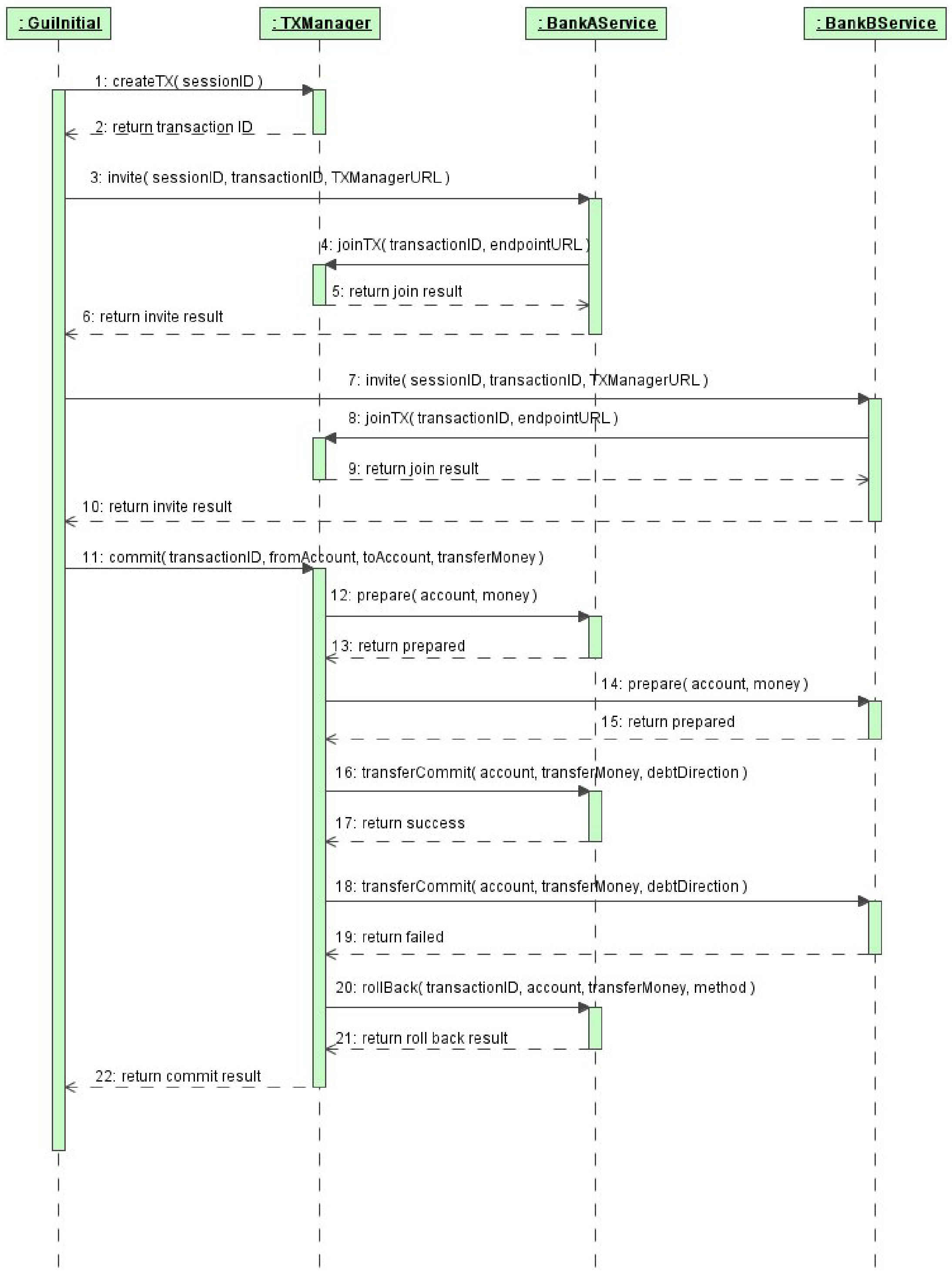

: GuiInitial
: TXManager
: BankAService
: BankBService
1: createTX(sessionID)
2: return transaction ID
3: invite(sessionID, transactionID, TXManagerURL)
4: joinTX(transactionID, endpointURL)
5: return join result
6: return invite result
7: invite(sessionID, transactionID, TXManagerURL)
8: joinTX(transactionID, endpointURL)
9: return join result
10: return invite result
11: commit(transactionID, fromAccount, toAccount, transferMoney)
12: prepare(account, money)
13: return prepared
14: prepare(account, money)
15: return prepared
16: transferCommit(account, transferMoney, debtDirection)
17: return success
18: transferCommit(account, transferMoney, debtDirection)
19: return success
20: finishTransaction(transactionID)
21: finishTransaction(transactionID)
22: return commit result

Figure 36 Sequence Diagram of Transferring Money from Bank A to Bank B

The process of transferring money successfully from Bank A to Bank B has been introduced in Figure 36. From Figure 36 it can be seen that the transaction manager abides by the two-phase commit protocol to control the transaction. In other words, the transaction manager dispatches money transferal messages to all the participants, only when it receives the “prepared” responses from all the participants, which means that all the participants agree to go ahead. During the money transferal procedure, if Bank A transfers money successfully and Bank B fails due to system problems, the transaction manger will ask Bank A Web service to rollback what has been done in Bank A Web service. The rollback procedure is shown in Figure 37. In step 19 of Figure 37, the transaction manager receives the “failed” information from Bank B Web service, and then it invokes the rollback function in Bank A Web service with the transaction ID to ask Bank A Web service to recover the original state.



: GuiInitial
: TXManager
: BankAService
: BankBService
1: createTX(sessionID)
2: return transaction ID
3: invite(sessionID, transactionID, TXManagerURL)
4: joinTX(transactionID, endpointURL)
5: return join result
6: return invite result
7: invite(sessionID, transactionID, TXManagerURL)
8: joinTX(transactionID, endpointURL)
9: return join result
10: return invite result
11: commit(transactionID, fromAccount, toAccount, transferMoney)
12: prepare(account, money)
13: return prepared
14: prepare(account, money)
15: return prepared
16: transferCommit(account, transferMoney, debtDirection)
17: return success
18: transferCommit(account, transferMoney, debtDirection)
19: return failed
20: rollBack(transactionID, account, transferMoney, method)
21: return roll back result
22: return commit result

Figure 37 Sequence Diagram of Transaction Rollback Procedure

5.4 System Deployment

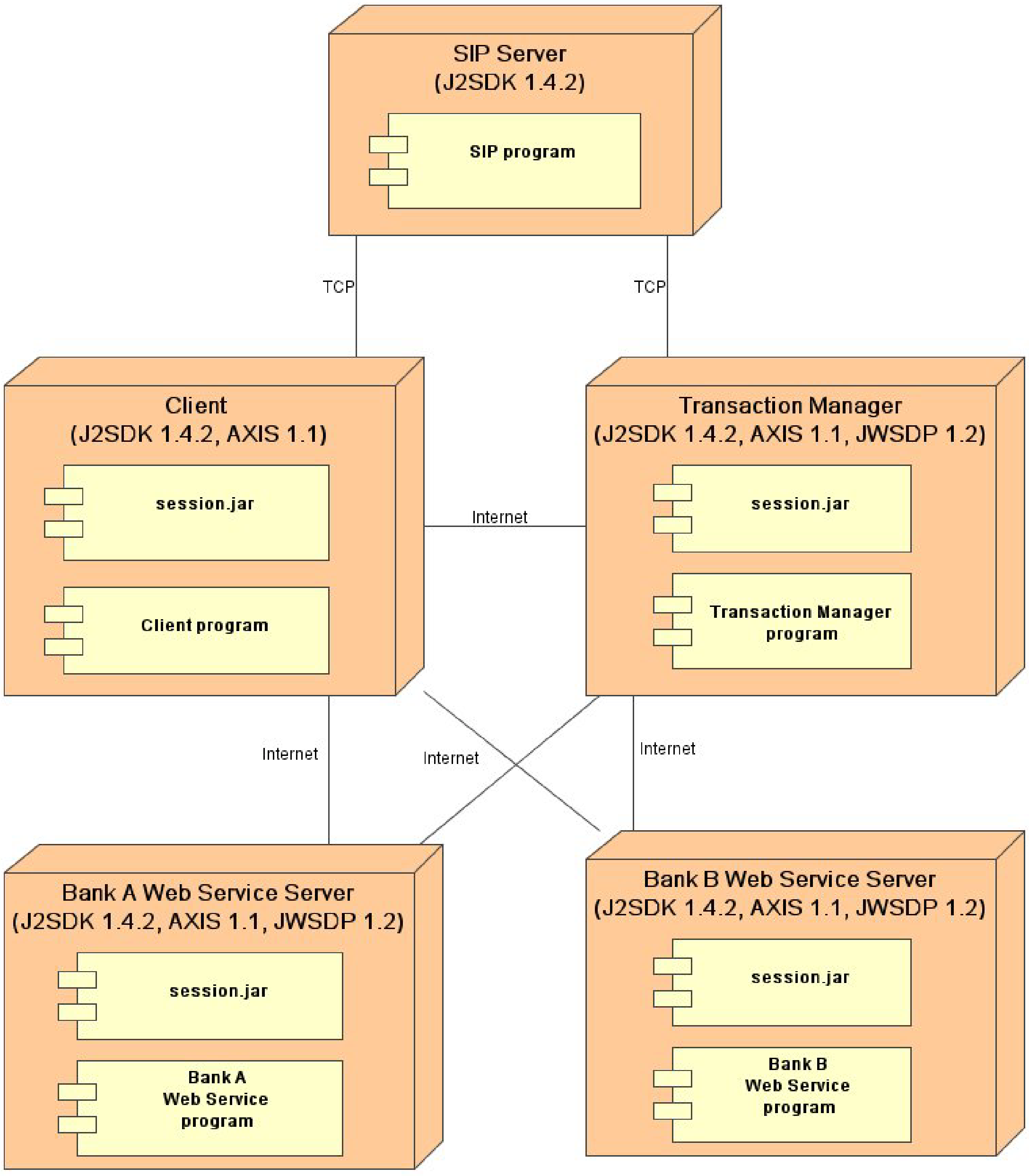
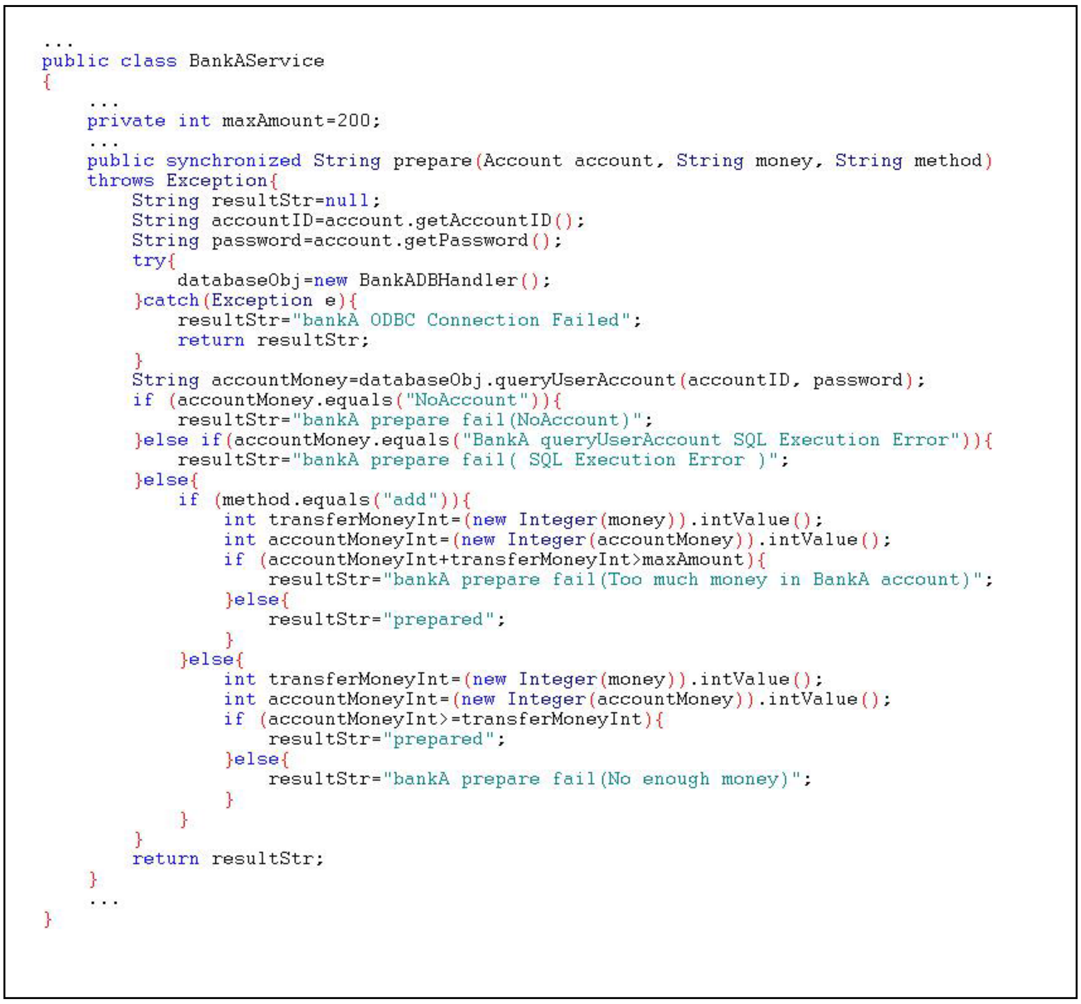

SIP Server
(J2SDK 1.4.2)
SIP program
TCP
TCP
Client
(J2SDK 1.4.2, AXIS 1.1)
session.jar
Client program
Internet
Transaction Manager
(J2SDK 1.4.2, AXIS 1.1, JWSDP 1.2)
session.jar
Transaction Manager program
Internet
Internet
Internet
Bank A Web Service Server
(J2SDK 1.4.2, AXIS 1.1, JWSDP 1.2)
session.jar
Bank A Web Service program
Bank B Web Service Server
(J2SDK 1.4.2, AXIS 1.1, JWSDP 1.2)
session.jar
Bank B Web Service program

Figure 38 Deployment diagram of the E-banking Web Services System

The deployment diagram of the E-banking Web Services System is shown in Figure 38. All computers in the system need to support J2SDK 1.4.2. The *session* jar file and AXIS 1.1 API are deployed in client computer, transaction manager, Bank A Web Service Server and Bank B Service Server. In order to run a Web service server, the JWSDP 1.2 is installed on transaction manager, Bank A Web service Server and Bank B Web service Server. The communication between SIP server and client, and between SIP and transaction manager is through TCP, and the communication among the client, transaction manager, Bank A Web service and Bank B Web Service is through the Internet.

5.5 System Testing

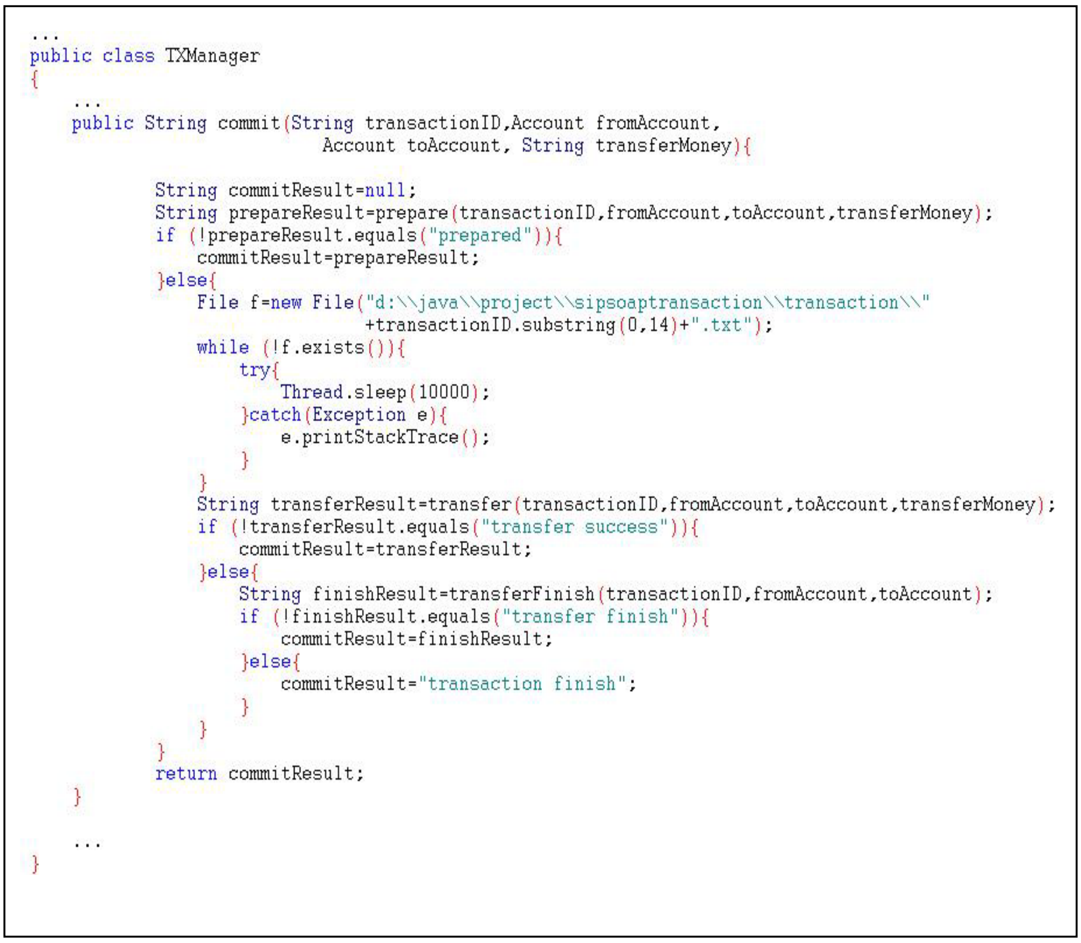
In order to know whether the rollback function in the transaction manager works well a scenario is simulated to test it. There are two assumptions in this scenario. The first is that the money amount in every account has a maximum value in both of the bank Web services. It is assumed that every account in Bank A can only have maximum $200, and in Bank B $100. For example, if John has an account in Bank B with $80, and he wants to transfer $30 from his Bank A account to his Bank B account, he will get a "Bank B prepare fail (Too much money in Bank B account)" information because $80 plus $30 is $110, which is greater than maximum account value $100 in Bank B. The Bank A Web service *prepare()* method code is illustrated in Figure 39.



```
...
public class BankAService
{
    ...
    private int maxAmount=200;
    ...
    public synchronized String prepare(Account account, String money, String method)
    throws Exception{
        String resultStr=null;
        String accountID=account.getAccountID();
        String password=account.getPassword();
        try{
            databaseObj=new BankADBHandler();
        }catch(Exception e){
            resultStr="bankA ODBC Connection Failed";
            return resultStr;
        }
        String accountMoney=databaseObj.queryUserAccount(accountID, password);
        if (accountMoney.equals("NoAccount")){
            resultStr="bankA prepare fail(NoAccount)";
        }else if(accountMoney.equals("BankA queryUserAccount SQL Execution Error")){
            resultStr="bankA prepare fail( SQL Execution Error )";
        }else{
            if (method.equals("add")){
                int transferMoneyInt=(new Integer(money)).intValue();
                int accountMoneyInt=(new Integer(accountMoney)).intValue();
                if (accountMoneyInt+transferMoneyInt>maxAmount){
                    resultStr="bankA prepare fail(Too much money in BankA account)";
                }else{
                    resultStr="prepared";
                }
            }else{
                int transferMoneyInt=(new Integer(money)).intValue();
                int accountMoneyInt=(new Integer(accountMoney)).intValue();
                if (accountMoneyInt>=transferMoneyInt){
                    resultStr="prepared";
                }else{
                    resultStr="bankA prepare fail(No enough money)";
                }
            }
        }
        return resultStr;
    }
    ...
}
```

Figure 39 Bank A's Web Service Code Segment

The second assumption is that after the transaction manager gets the "prepared" information from both bank Web services, it also needs to locate a text file, which is named by first fourteen digits of the session ID and then it can commit the transferring money action. For instance, John has established a session with the SIP server and has got the session ID which is 20040919142242@192.168.0.1. Then he creates a transaction with the transaction manager and wants to transfer money between two banks. After the transaction manager gets the "prepared" messages from both of the bank Web services, there should be a text file named "20040919142242.txt" that can be located by the transaction manager in order to transfer the money. Figure 40 displays the *commit()* method code in the transaction manager.



```
...
public class TXManager
{
    ...
    public String commit(String transactionID,Account fromAccount,
                         Account toAccount, String transferMoney){

            String commitResult=null;
            String prepareResult=prepare(transactionID,fromAccount,toAccount,transferMoney);
            if (!prepareResult.equals("prepared")){
                commitResult=prepareResult;
            }else{
                File f=new File("d:\\java\\project\\sipsoaptransaction\\transaction\\"
                                +transactionID.substring(0,14)+".txt");
                while (!f.exists()){
                    try{
                        Thread.sleep(10000);
                    }catch(Exception e){
                        e.printStackTrace();
                    }
                }
                String transferResult=transfer(transactionID,fromAccount,toAccount,transferMoney);
                if (!transferResult.equals("transfer success")){
                    commitResult=transferResult;
                }else{
                    String finishResult=transferFinish(transactionID,fromAccount,toAccount);
                    if (!finishResult.equals("transfer finish")){
                        commitResult=finishResult;
                    }else{
                        commitResult="transaction finish";
                    }
                }
            }
            return commitResult;
    }

    ...
}
```

Figure 40 Transaction Manager Code Segment

In this scenario, Bank A and Bank B accounts detail are given as below: (See Table 6 and Table 7)

Account Number	Password	Account Balance
123 (John)	123	100
234 (Kate)	234	120

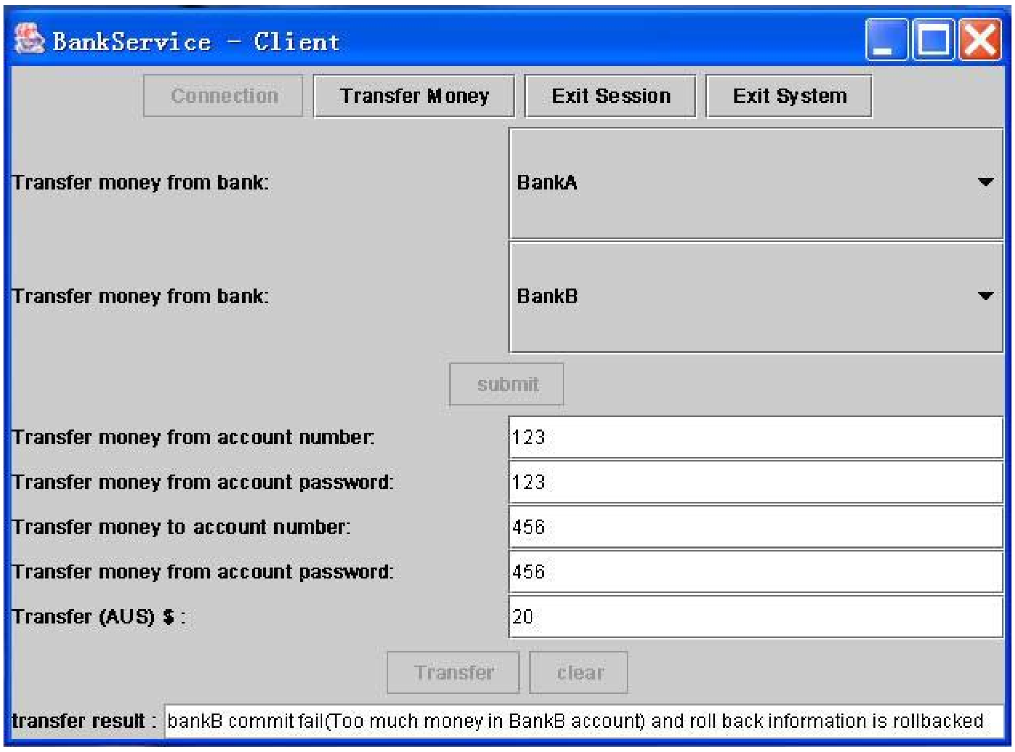
Table 6 Bank A Accounts Detail

Account Number	Password	Account Balance
345	345	40
456 (Joint)	456	60

Table 7 Bank B Accounts Detail

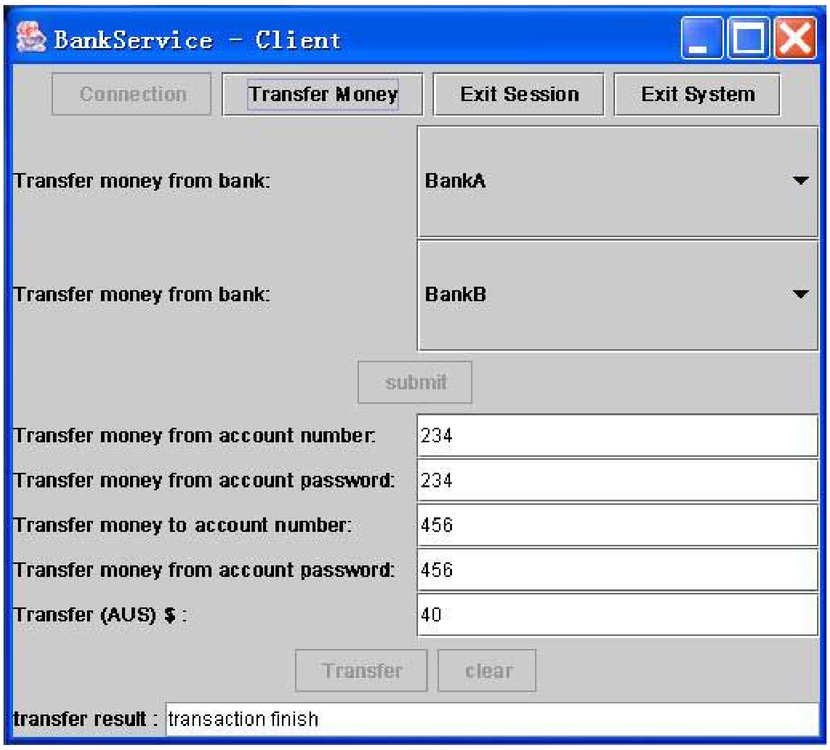
Two users, John and Kate, take part in this scenario. John withdraws $20 from account number 123 at Bank A, while Kate withdraws $40 from account number 234 at Bank A. They want to transfer the money to account number 456 at Bank B. First John's transaction starts, and then Kate's. But Kate's session text file is created before John's. In this situation, the transaction result of Kate's money transferal should be successful because the result will be within the $100 limit of account 456. On the other hand John's transaction should fail because Kate's transaction preceded John's, and at that time of John's transaction the money in account number 456 was already $100. Although the transaction manager has got the "prepared" message from both bank Web services for John's transaction, and the money transferal action can be accomplished in Bank A, the account number 456 in Bank B cannot accept the deposit. Hence, the transaction manager asks Bank A to rollback the money into account number 123.

The testing result is similar to the above. Figure 41 and Figure 42 show John's and Kate's money transferal transaction results respectively.



Transfer money from bank:	BankA
Transfer money from bank:	BankB
Transfer money from account number:	123
Transfer money from account password:	123
Transfer money to account number:	456
Transfer money from account password:	456
Transfer (AUS) $:	20
transfer result :	bankB commit fail(Too much money in BankB account) and roll back information is rollbacked

Figure 41 John's Transaction Result


BankService - Client

Connection | Transfer Money | Exit Session | Exit System

Transfer money from bank: BankA

Transfer money from bank: BankB

submit

Transfer money from account number: 234

Transfer money from account password: 234

Transfer money to account number: 456

Transfer money from account password: 456

Transfer (AUS) $: 40

Transfer | clear

transfer result : transaction finish

Figure 42 Kate's Transaction Result

The closing balance of account number 456 in Bank B is $100, and in Bank A the balance of account number 123 is 100, and the balance of account number 234 is $80.

This scenario result demonstrates that the transaction manager in the E-banking Web Service System supports the two-phase commit protocol to control the transaction. Moreover, it is able to handle multiple transactions and keep the atomicity and consistency of the transactions.

5.6 Summary

This chapter has presented the way use of SIP to manage transactions for Web services, and the E-banking Web Services System has been developed as a demonstration project. The transaction manager, which is able to coordinate the transaction for all participants, is also a Web service. It conforms to the two-phase commit model which means that all participants have to agree on the transaction, only then can the transaction be committed. During the implementation of the transaction, if any participants cannot accomplish the action successfully, the participants need to rollback to the original state. A unique transaction ID is used to label every transaction created in the transaction manager. All other participants can use the transactions ID to communicate with the transaction manager, which enables data state consistency in all participants.

Although the transaction management for simple Web services has been accomplished, part of the implementation of the transaction manager would need to be changed for diverse business systems in order to coordinate different Web services. The security and reliability issues also need to be considered in future work.

Chapter 6 Evaluation and Conclusion

This thesis has proposed the mechanisms of using SIP to manage session and transaction for Web services. Two demonstration systems have been developed to implement the mechanisms. This chapter first reviews and summarizes the thesis in section 6.1 and then presents the answers to the research questions proposed in Chapter 1 in section 6.2. Section 6.3 evaluates the proposed mechanisms. After that, the results of the hypotheses proposed in Chapter 3 are given in section 6.4. Finally, the future work will be outlined in section 6.5.

6.1 Research Summary

This thesis has offered new mechanisms that support session and transaction management on Web services by using a standard session control protocol SIP.

Chapter 2 introduced the background knowledge for this research. Web services, as one of the most popular technical innovations discussed today, have attracted considerable attention in industry. Among the three main technologies, SOAP, WSDL, and UDDI, behind Web services, SOAP possesses an important function. SOAP messages can be exchanged over a variety of protocols, but SOAP can also accommodate various programming languages. However, SOAP is a connectionless protocol, which means there is no session control on SOAP. With Web services applied widely to E-commerce applications, the session state is required to be maintained. As discussed in chapter 2, traditional session management techniques are not suitable for managing sessions of SOAP messages, and some new session management techniques also have their limitations. Also in Chapter 2, some background on the two IETF standardized protocols SIP and SDP was provided.

Chapter 3 first described the status of Web services transaction management and then provided the hypotheses of this research. Following that, the implementation gaps of Web services session and transaction managements were discussed. Finally, this chapter proposed the objectives of this thesis. These were developing a new session management mechanism for Web services by using a standard session control protocol SIP, and then based on this mechanism providing a simple Web service transaction management solution. In order to demonstrate the mechanisms, two systems were discussed: an Online Video Shopping Web Service System and an E-banking Web Service System.

Chapter 4 presented the analysis, design and implementation of the Online Video Shopping Web Service System. In this system, a SIP server was used to control the session setting up and terminating for the Web service. The session ID generated by SIP was inserted into a SOAP header element and delivered between the Web services client and server. There was a session table maintained by the service server, which could be used to keep all session states on the Web service. It is important to emphasize here that using this proposed architecture to implement Web services system, the SOAP server needs to provide an interface or function for programmers to accomplish the session information exchange between SOAP messages and back-end Web service programs, as with AXIS.

Based on the first system, the E-banking Web Services System was implemented in Chapter 5. The architecture of this system was very similar to the previous one. The difference was that a transaction manager was implemented as a kind of Web service to communicate with the SIP server and to control the transaction for the Web service client and the two bank Web services. Although this system provided a simple service, i.e., transferring money between two banks, it conformed to the two-commit protocol to manage the transaction and was able to keep the atomicity and consistency properties of ACID transaction.

6.2 Research Questions

The research questions were introduced in Chapter 1. Following is a summary answer to each question as proved in the thesis.

Question 1 is whether SIP can be used to manage sessions and transactions for Web services.
Answer: Yes. In this thesis, two Web service systems have been implemented to demonstrate how to use SIP to manage sessions and transactions for Web services.

Question 2 is how to inform Web services the beginning and the ending of a Web service session by using SIP.
Answer: The implementation of the Online Video Shopping Web Service System provides the answer for this question.

The procedure for a Web service to know the beginning of a session is: First a session is established between a client application and a SIP server. The client application then gets a session ID. When the Web service server receives a SOAP request which contains the session ID, it will check the session ID with the SIP server and will know whether the session has been established or not.

The procedure for a Web service to know the ending of a session is: When a SIP server receives a BYE message from a client application, it will inform the Web service server by sending a session terminating message which contains the session ID. As a result, the Web service knows that the session has terminated.

Question 3 is how to manage multiple sessions and transactions on the Web service server.
Answer: In order to manage multiple sessions and transactions on the Web service server, a *SessionTable* class is used in both of the demonstration systems. It keeps a session ID or a transaction ID and their relative data as a key – value pair, (for

example, in the Online Video Shopping Web Service System the session table keeps the session ID and shopping cart as a key – value pair) as a result, the Web service server can handle multiple sessions and transactions by administrating the session table.

Question 4 is how to keep the session state between the client application and the Web service server during communication.
Answer: After a client application establishes a session with a SIP server, it will get a session ID. The set up session ID will be contained in the SOAP messages sent between the client application and the Web service server. Thus, the session state can be kept during communication.

6.3 Evaluation

6.3.1 Session Management Mechanism Evaluation

The session management mechanism proposed in this thesis involves adding session information in the SOAP header element and using SIP to manage the session state for Web services. This mechanism possesses three advantages compared to similar mechanisms introduced in chapter 2, for example, Jeckle's and Microsoft Corporation's SOAP session solutions. These three advantages are described as follows:

1. Using standard protocol to generate session ID

   The session information added in the SOAP header element can vary in Jeckle's and Microsoft Corporation's SOAP session solutions. If using different ways to identify sessions on Web services, especially in E-commerce, it will be hard to integrate them across enterprises. Since Web services are based on standard protocols, such as UDDI, WSDL and SOAP, to support the interoperable interaction between different applications over a network, a standard way of describing the session ID is a suitable solution for accomplishing session management.

The advantage of the proposed mechanism is that SDP, which is developed by IETF MMUSIC for describing multimedia sessions, and ISO 8601 Date/Time Representations are used to generate the session ID. The format of the session ID is:

YYYYMMDDhhmmss@<SIP client IP address>

Using this standard manner to identify a session makes it easier to integrate Web services across enterprises.

2. Using standard protocol to manage the session for Web services

In Jeckle's SOAP session solution, the session management mechanism is left to Web developers. This will also cause an integration problem between Web services. The Microsoft Corporation's solution for managing sessions on Web services is similar to the HTTP cookie mechanism. This means that the Web developers have to make sure that the client application treats the <BeginSession>, <Session> and <EndSession> tags as HTTP cookies and knows when and how to send those three tags to the Web server. If the Web server crashes, it will be difficult to recover the session state.

The second advantage of the current mechanism is that as SIP, a standard application-layer session control protocol, is used to manage sessions for Web services, the integration problem can be easily solved according to SIP specification. In addition, the SIP server can be located in PCs separated from the Web server. If the Web server crashes, the session state can be recovered from SIP server. Moreover, using SIP to manage the session state enhances the extensibility to Web services as more Web services can be easily added into the demonstration system.

3. Extensibility on security issue

The security issue is always important for the E-commerce Web services, as customers need to input sensitive information such as credit card numbers in order to access services. However, neither Jeckle nor Microsoft Corporation mentions the solution for security in their SOAP session management mechanisms.

Although in the demonstration project there is no code for implementing the security, the SIP specification can be the reference for adding the security solution to the mechanism. Actually, the SIP specification defines the communication encryption and authentication and also provides security solutions for registration, inter-domain requests, peer-to-peer requests, and denial-of-service protection. Therefore, the security issue can be easily extended in the proposed mechanism. This is the third advantage of the current mechanism.

6.3.2 Transaction Management Mechanism Evaluation

Based on the session management mechanism, this thesis has proposed a simple transaction management mechanism on Web services. In this mechanism, SIP is still used to manage the session state and a transaction manager is introduced to manage transactions among participants.

As introduced in chapter 3, the WS-CAF [48] and WS-CDL [52] simply provide implementation guidelines for Web services transaction management. In fact, the design and implementation of transaction management depend on the business work flow. For example, in the E-banking Web service project, the implementation of the transaction manager is based on a money transferal work flow.

The advantages of the proposed transaction mechanism are: firstly, the transaction manager is a Web service which needs to communicate with the SIP server to control transactions among joined parties. It can be changed easily according to different businesses. Secondly, as SIP is still used to manage the session state, security issue can be addressed in this mechanism according to SIP specification. Thirdly, this mechanism conforms to two-phase commit protocol which enables data state consistency in all participants. Finally, this mechanism provides great extensibility for more Web services to be added in.

6.4 Hypotheses Results

Based on the evaluation in the last section, the results of the hypotheses proposed in Chapter 3 are:

Hypothesis 1: The standard session control protocol — SIP can be used to manage sessions and transactions for Web services.

Result: Correct. It has been proved by the accomplishment of the two demonstration systems.

Hypothesis 2: Using SIP to manage sessions for Web services is more elegant and general than Jeckle's and Microsoft Corporation's solutions.

Result: Correct. SIP not only provides a standard manner for identifying a session, but also solves the integration problem in Jeckle's and Microsoft Corporation's solutions.

Hypothesis 3: Using SIP to manage transactions for Web services can keep the atomicity and consistency of the transactions.

Result: Correct. This can be proved by the system testing in Chapter 5.

Hypothesis 4: Using SIP to manage sessions and transactions for Web services provides an easier solution on security issue than other solutions proposed so far.

Result: Correct. As mentioned in section 6.3, SIP specification has provided clear definitions for implementing security.

6.5 Future Work

Instead of inventing a new protocol, this thesis uses an existing standard protocol SIP to manage sessions and transactions for Web services. However, the security issue has not been considered in these mechanisms. As mentioned in section 6.1, SIP specification mentions various security solutions. Some of these could be added into the mechanisms in future research.

In the Web service systems, it is assumed that the service customer already knows the location of the Web service provider. Hence, the service registry and discovery procedures are omitted. In order to provide a complete Service-Oriented Architecture implementaion, a UDDI server could be added to a future system.

The transaction management is still under development because Web services are loosely-coupled, have long transaction durations and complex interactions between multiple components. This thesis has provided a simple solution for transaction management on Web services. Hopefully, it will bring about new implementation possibilities for managing Web service transactions in the future.

References

1. **Ray, R. J., and Kulchenko, P.**, *Programming Web Serivices with Perl*, O'Reilly&Associates,Inc. (2003).
2. **Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D.**, *Web Services Architecture*, (2004). Available from http://www.w3.org/TR/ws-arch/ [Accessed: Oct. 22, 2004]
3. **Irani, R., and Basha, S. J.**, *AXIS: The Next Generation of Java SOAP*, Wrox Press Ltd (2002).
4. **UDDI**, *UDDI technical white paper*, (September 6, 2000). Available from http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf [Accessed: Oct. 22, 2004]
5. **Newmarch, J.**, *Web services*, (Oct 22, 2003). Available from http://jan.netcomp.monash.edu.au/webservices/tutorial.html [Accessed: Oct. 22, 2004]
6. **Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S.**, *Web Services Description Language (WSDL) 1.1*, (15 March, 2001). Available from http://www.w3.org/TR/wsdl [Accessed: Oct. 22, 2004]
7. **Looker, N., and Xu, J.**, *Assessing the dependability of SOAP RPC-based Web services by fault injection*, *Object-Oriented Real-Time Dependable Systems, 2003. Proceedings. Ninth IEEE International Workshop,* Vol, Iss, p. 163 - 170, (2003).
8. **Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., and Nielsen, H. F.**, *SOAP Version 1.2 Part 1: Messaging Framework*, (2003). Available from http://www.w3.org/TR/2003/REC-soap12-part1-20030624/ [Accessed: Setp. 30, 2004]
9. **Jepsen, T.**, *SOAP cleans up interoperability problems on the Web*, *IEEE IT Professional,* Vol: 3, Iss: 1, p. 52 - 55, (2001).

10. **Mitra, N.**, *SOAP Version 1.2 Part 0: Primer*, (24 June, 2003). Available from http://www.w3.org/TR/2003/REC-soap12-part0-20030624/ [Accessed: Sept. 30, 2004]

11. **Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., and Nielsen, H. F.**, *SOAP Version 1.2 Part 2: Adjuncts*, (24 June, 2003). Available from http://www.w3.org/TR/2003/REC-soap12-part2-20030624/ [Accessed: Sept. 30, 2004]

12. **Ts'o, T., and Chiu, A.**, *ONC Remote Procedure Call (oncrpc)*, (16 Jan., 2001). Available from http://www.ietf.org/html.charters/oncrpc-charter.html [Accessed: 20 Oct., 2004]

13. **Chester, T. M.**, *Cross-platform integration with XML and SOAP*, *IEEE IT Professional,* Vol: 3, Iss: 5, p. 26 - 34, (2001).

14. **Tsenov, M.**, *Application of SOAP protocol in E-commerce Solution*, *2002 First International IEEE Symposium,* Vol: 3, Iss, p. 59 - 62, (2002).

15. **Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S.**, *Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI*, *Internet Computing, IEEE,* Vol: 6, Iss: 2, p. 86 -93, (2002).

16. **Liu, C. C., and Chuang, Y. D.**, *Remote Data Access Scheme Support for Wireless Access to an Online Teaching System Using SOAP Technology*, *IEEE Telecommunications,* Vol: 2, Iss, p. 1717 - 1722, (2003).

17. **UPnP**, *Understanding Universal Plug and Play (White Paper)*, (2002). Available from http://www.upnp.org/download/UPNP_UnderstandingUPNP.doc [Accessed: Oct. 22, 2004]

18. **Postel, J.**, *RFC: 793 TRANSMISSION CONTROL PROTOCOL*, (September, 1981). Available from http://www.faqs.org/rfcs/rfc793.html [Accessed: 20 Oct., 2004]

19. **Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T.**, *Hypertext Transfer Protocol -- HTTP/1.1*, (June, 1999).

Available from http://www.w3.org/Protocols/rfc2616/rfc2616.html [Accessed: 20 Oct., 2004]

20. **Jeckle, M. C.**, *Adhere to Session-Oriented Communication Principles*, (2002). Available from http://www.jeckle.de/files/ssgrr2002.pdf [Accessed: 5 Oct., 2003]

21. **Patterson, J. F., Hill, R. D., Rohall, S. L., and Meeks, S. W.**, *An Architecture for Synchronous Multi-user Applications*, *Computer Supported Cooperative Work, Proceedings of the 1990 ACM conference on Computer-supported cooperative work,* Vol, Iss, p. 317 - 328, (1990).

22. **Hughes, M., Shoffner, M., and Hamner, D.**, *Java Network Programming second edition*, Manning Publications Co. (1999).

23. **Schildt, H.**, *The Complete Reference Java 2 Fifth Edition*, Brandon A. Nordin (2002).

24. **Boutell, T.**, *CGI Programming in C& Perl*, Kim Fryer (1996).

25. **Chan, H., Lee, R., Dillon, T., and Chang, E.**, *E-Commerce Fundamentals and Applications*, John Wiley & Sons, Inc. (2001).

26. **Peng, W. H., and Clisna, J.**, *HTTP cookies- a Promising Technology*, *Online Information Review,* Vol: 24, Iss: 2, p. 150 - 153, (2000).

27. **Powell, M.**, (2002). Available from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnservice/html/service08062002.asp [Accessed: 5 Oct., 2003]

28. **Berghel, H.**, *Hijacking the Web----Cookies revisited: Continue the dialog on personal security and underling privacy issue*, *Communication of the ACM,* Vol: 45, Iss: 4, p. 23 - 27, (2002).

29. **Newmarch, J.**, *A Critique of Web Services*, (2004). Available from http://jan.netcomp.monash.edu.au [Accessed: 26 Oct., 2004]

30. **Corporation, M.**, *DMSL SOAP session support*, (2003). Available from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dsml/dsml/dsml_soap_session_support.asp [Accessed: 5 Oct., 2004]

31. **Cai, H. L., W., Yang, B., and Tang, L. H.**, *Session Initiation Protocol and*

Web Services for Next Generation Multimedia Applications, *Multimedia Software Engineering, 2002. Proceedings of the IEEE Fourth International Symposium on Multimedia Software Engineering,* Vol, Iss, p. 70 - 80, (2002).

32. **Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E.**, *RFC 3261 - SIP: Session Initiation Protocol*, (Jun., 2002). Available from http://www.faqs.org/rfcs/rfc3261.html [Accessed: 5 Oct., 2003]

33. **Postel, J.**, *RFC 768 - User Datagram Protocol*, (28 August, 1980). Available from http://www.faqs.org/rfcs/rfc768.html [Accessed: 20 Oct., 2004]

34. **Dierks, T., and Allen, C.**, *RFC 2246 - The TLS Protocol Version 1.0*, (January, 1999). Available from http://www.faqs.org/rfcs/rfc2246.html [Accessed: 20 Oct., 2004]

35. **Johnston, A. B.**, *SIP Understanding the Session Initiation Protocol*, ARTECH HOUSE, INC. (2004).

36. **Handley, M., and Jacobson, V.**, *RFC 2327 - SDP: Session Description Protocol*, (April, 1998). Available from http://www.faqs.org/rfcs/rfc2327.html [Accessed: 20 Oct., 2004]

37. **Handley, M., Perkins, C., and Whelan, E.**, *RFC 2974 - Session Announcement Protocol*, (October, 2000). Available from http://www.faqs.org/rfcs/rfc2974.html [Accessed: 20 Oct., 2004]

38. **Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V.**, *RFC 3550 - RTP: A Transport Protocol for Real-Time Applications*, (July, 2003). Available from http://www.faqs.org/rfcs/rfc3550.html [Accessed: 20 Oct., 2004]

39. **Schulzrinne, H., Rao, A., and Lanphier, R.**, *RFC 2326 - Real Time Streaming Protocol (RTSP)*, (April, 1998). Available from http://www.faqs.org/rfcs/rfc2326.html [Accessed: 20 Oct., 2004]

40. **Postel, J.**, *RFC 791 - Internet Protocol*, (September, 1981). Available from http://www.faqs.org/rfcs/rfc791.html [Accessed: 20 Oct., 2004]

41. **Xia, N.**, *A Modem Based Lightweight VoIP System Using SIP*, in *School of*

Network Computing, Monash University, Melbourne (January, 2004).

42. **Camarillo, G.**, *SIP Demystified*, McGraw-Hill Companies, Inc. (2002).

43. **Anderson, R.**, *Web Services Management - Plotting a Course for Success (Slow adoption requires careful planning)*, (April 5, 2004). Available from http://www.sys-con.com/story/?storyid=44359&DE=1 [Accessed: 26 Aug., 2004]

44. **serviceoriented.org**, *ACID Transactions*, (Available from http://www.serviceoriented.org/acid_transactions.html [Accessed: 17 Aug., 2004]

45. **NuSphere.com**, *ACID Transactions*, (2002). Available from http://www.nusphere.com/products/library/acid_transactions.htm [Accessed: 17 Aug., 2004]

46. **Padmanabhuni, S.**, *Web services transactions standards: Core requirements*, (10 Jul., 2003). Available from http://searchwebservices.techtarget.com/originalContent/0%2C289142%2Csid26_gci913977%2C00.html [Accessed: 26 Aug., 2004]

47. **Siegrist, J. L.**, *Transaction Management Issues and Recommendations for Developing Extranet - Based Web Services for the Oil Storage Company*, (April/May, 2004). Available from http://www.jenikya.com/text/articles/transwebservices.pdf [Accessed: 19 Aug., 2004]

48. **Bunting, D., Chapman, M., Hurley, O., Little, M., Mischkinsky, J., Newcomer, E., Webber, J., and Swenson, K.**, *Web Services Composite Application Framework (WS-CAF) Ver1.0*, (July 28, 2003). Available from http://developers.sun.com/techtopics/webservices/wscaf/primer.pdf [Accessed: 12 Nov., 2004]

49. **Bunting, D., Chapman, M., Hurley, O., Little, M., Mischkinsky, J., Newcomer, E., Webber, J., and Swenson, K.**, *Web Services Context (WS-Context) Ver1.0*, (July 28, 2003). Available from http://developers.sun.com/techtopics/webservices/wscaf/wsctx.pdf [Accessed:

10 Nov., 2004]

50. **Bunting, D., Chapman, M., Hurley, O., Little, M., Mischkinsky, J., Newcomer, E., Webber, J., and Swenson, K.**, *Web Services Coordination Framework (WS-CF) Ver1.0*, (July 28, 2003). Available from http://developers.sun.com/techtopics/webservices/wscaf/wscf.pdf [Accessed: 10 Nov., 2004]

51. **Bunting, D., Chapman, M., Hurley, O., Little, M., Mischkinsky, J., Newcomer, E., Webber, J., and Swenson, K.**, *Web Services Transaction Management (WS-TXM) Ver1.0*, (July 28, 2003). Available from http://developers.sun.com/techtopics/webservices/wscaf/wstxm.pdf [Accessed: 19 Aug., 2004]

52. **Kavantzas, N., Burdett, D., and Ritzinger, G.**, *Web Services Choreography Description Language Version 1.0*, (27 April, 2004). Available from http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/ [Accessed: 26 Aug., 2004]

53. *ISO 8601 Date/Time Representations*, (Available from http://www.mcs.vuw.ac.nz/technical/software/SGML/doc/iso8601/ISO8601.html [Accessed: 19 Dec., 2004]

54. **Kuhn, M.**, *A Summary of the International Standard Date and Time Notation*, (19 Dec., 2004. Available from http://www.cl.cam.ac.uk/~mgk25/iso-time.html [Accessed: 21 Dec., 2004]

Appendix A Glossary

HTML	Hypertext Markup Language
UDDI	Universal Description, Discovery and Integration
WSDL	Web Services Description Language
SOAP	Simple Object Access Protocol
W3C	World Wide Web Consortium
XML	Extensible Markup Language
IETF	Internet Engineering Task Force
SIP	Session Initialtion Protocol
SDP	Session Description Protocol
RPC	Remote Procedure Call
COM	Component Object Model
DCOM	Distributed Component Object Model
CORBA	Common Object Request Broker Architecture
IDL	Interface Definition Language
ORB	Object Request Broker
HTTP	Hypertext Transfer Protocol
SOA	Service-Oriented Architecture
RMI	Remote Method Invocation
IIOP	Internet Interoperable ORB Protocol
ONC RPC	Open Network Computing Remote Procedure Call
LAN	Local Area Network
WAN	Wide Area Network
SMTP	Simple Mail Transfer Protocol
FTP	File Transfer Protocol
WAP	Wireless Application Protocol
UPnP	Universal Plug and Play
TCP	Transmission Control Protocol

OSI	Open Systems Interconnect
IP	Internet Protocol
CGI	Common Gateway Interface
OS	Operation System
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
DSML	Directory Services Markup Language
UDP	User Datagram Protocol
TLS	Transmission Layer Security
SAP	Session Announcement Protocol
RTP	Real-Time Transport Protocol
RTSP	Real-Time Streaming Protocol
UA	User Agent
UAS	User Agent Server
UAC	User Agent Client
MMUSIC	Multiparty Multimedia Session Control
MIME	Multipurpose Internet Mail Extensions
ACID	Atomic, Consistent, Isolation, and Durable
WS-CAF	Web Services Composite Application Framework
WS-CTX	Web Services Context Service Specification
WS-CF	Web Services Coordination Framework Specification
WS-TXM	Web Services Transaction Management Specification
TX-ACID	ACID transactions model
TX-LRA	Transaction Long running action model
TX-BP	Transaction Business process transaction model
WS-CDL	Web Services Choreography Description Language
JSP	Java Server Pages
ASP	Active Server Pages
AXIS	Apache eXtensible Interaction System
JWSDP	Java^{TM} Web Services Developer Pack

VoIP	Voice over IP
ISO	International Organization for Standardization
WSDD	Web Service Deployment Descriptor