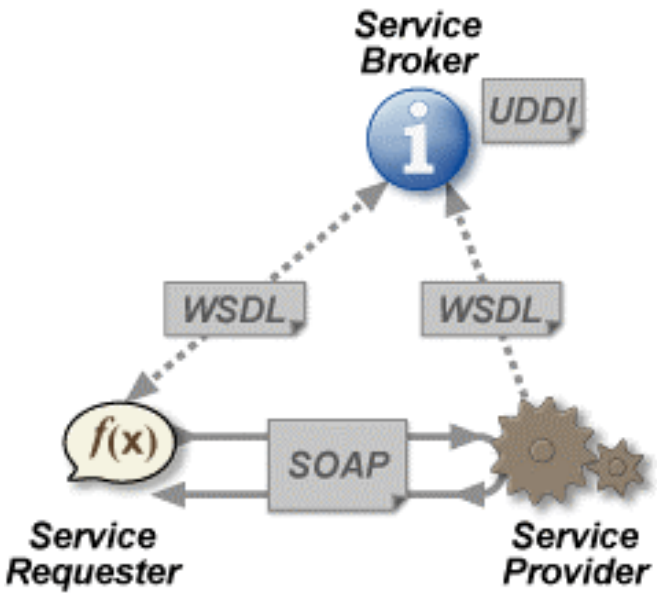
Web Services



Written by:
Ran Yang
Business Mathematics and Informatics
Free University, Amsterdam

Supervisor:
Prof. dr. R.D. van der Mei
Vrije Universiteit, Amsterdam

Preface

This paper has been written as the last assignment of my study Business Mathematics and Informatics (BMI) at the Free University Amsterdam. The objective is to investigate the available literatures in reference to a topic that should cover at least two out of the following areas: business management, mathematics or computer science.

Internet is playing an increasingly important role in daily life nowadays. People use Internet as a medium for on-line banking, to order the latest books or CDs on-line, to purchase a vacation package using an online travel agent, etc.. All these activities via Internet have to do with Web services. Although people use the term “Web services” very often, it is not always clear what Web services are and how Web services are built. In this paper, I tried to give answers to these two questions based of the available literature. In addition, I also investigated several QoS issues in Web services. The subject of this paper was set in consultation with prof.dr. R.D. van der Mei of the department of mathematics.

During the realisation of this paper, I have received excellent support from several people. I am indebted to all of them. Especially, I would like to express my appreciation and gratitude to my supervisor Rob for his constant guidance and advices. He provided me with useful input and moreover his infectious enthusiasm has certainly had its positive effect on my work.

Ran Yang

Delft, June 2005

Table of Content

1. Introduction

1.1 **Background and motivation**

The term “Web Services” is coined firstly by Microsoft in 2000 when they are introduced as a major component of .NET technology aimed at revolutionizing distributed computing. In 2002, Web services became a hot new technology that provides a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. Nowadays Web services have gained an enormous acceptance and a broad usage on different areas: e-commerce, search engines like Google, etc.. Nevertheless, for most people it is not always clear what Web services precisely are although they use them very often. It has been frequently considered that every service available through the Web is a Web service. This is a common mistake that leads to quite a lot of confusion when discussing Web services technology. In this report, we try to give the reader a correct understanding of Web services.

In a typical Web services scenario, a business application sends a request to a service at a given URL using the SOAP (simple object access protocol) over HTTP. The service receives the request, processes it, and returns a response. The time it takes is denoted as the Response Time (RT). The RT is one of the metrics included in a Web service’s QoS attributes [1, 2]. From a user’s point of view, the RT determines the perceived end-to-end browsing quality. A quality is often related to the cost. Shorter RT or faster service will be more expensive. For service providers it is important to find out how to satisfy the customer at the lowest price. Therefore, we need to construct a model for calculating the minimum cost while making the correspondent response time acceptable.

1.2 **Overview of this report**

In Chapter 2, we describe first what Web services are, what technologies are required for Web services, and how Web services are built. Chapter 3 discusses each Web service technology in detail. In Chapter 4, we describe QoS issues in Web services. Finally, we present our conclusions in Chapter 5.

2. Web services

2.1 **What are Web services**

The use of Web Services on the World Wide Web is expanding rapidly as the need for application-to-application communication and interoperability grows. These Web services provide a standard means of communication among different software applications, running on a variety of platforms and/or frameworks.

In the general meaning, Web services are services offered via the Web. But there is a difference between services in the software sense and services in the general sense, i.e., activities performed by a person or a company on behalf of another person of company. Take as examples bookstores, restaurants, or travel agencies. They all provide services. In some cases, a customer might even be able to obtain such services through the web server of the company. Strange as it might seem at first, this is not what Web services are about. A Web service is a software application with a published a stable programming interface, not a set of Web pages. According to [3], a Web service can be described as “a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols.” This definition states that Web services should be described and advertised so that it is possible to write clients that bind and interact with them. In other words, Web services are components that can be integrated into more complex distributed applications.

2.2 **An example of Web services**

An often-cited example of a Web service is that people purchase a vacation package using an online travel agent [4]. As Figure 2-1 shows, the travel site uses Web services applications for airline booking, hotel reservation, and car rental reservation. To locate the best prices on airline tickets, hotels, and rental cars, the agency will have to poll multiple companies, each of which likely uses different, incompatible applications for pricing and reservations. Web services aim to simplify this process by defining a standardized mechanism to describe, locate, and communicate with online applications. Essentially, each application becomes an accessible Web service component that is described using open standards. An online travel service could thus use the same Web services framework to locate and reserve package elements, as well as to lease Internet-based credit check and bank payment services on a pay-per-use basis to expedite fund transfers between customer, the travel agency, and the vendors.

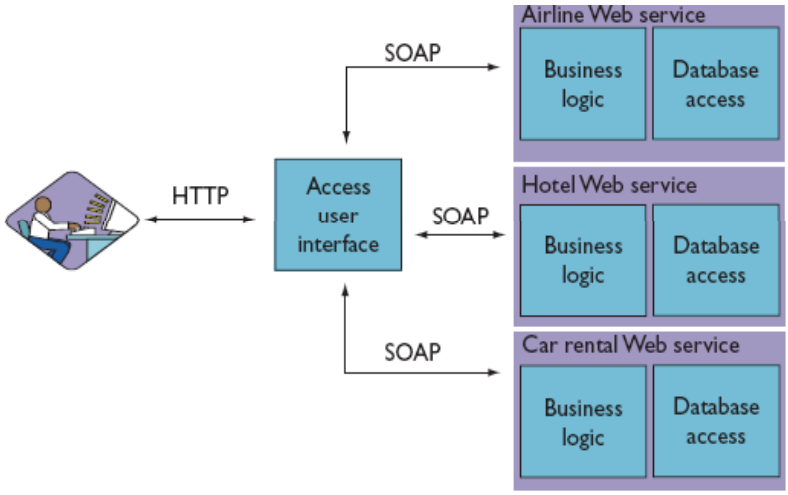


Figure 2-1: Under the Web services model, the travel site implements only the user interface, invoking airline, hotel and car rental reservation services via SOAP.

2.3 **Web services technologies**

After describing what Web services are, we now discuss what technologies are required for Web services. For more details, we refer to [5, 6].

To understand what technologies are required for Web services, we need to understand a typical Web service interaction first. The basic Web service architecture models the interactions between three roles: the service provider, service discovery agency, and service requestor. The interactions involve the publish, find, and bind operations. See Figure 2-2. In the first step, to make a Web service available to potential clients, a Web service provider must register information about itself, the services it offers and how to invoke them, in a Web Service Registry using universal description, discovery, and integration (UDDI). The description of services and how to invoke them is normally written in Web Service Description Language (WSDL). Secondly, a client who is looking for a service to meet its requirement searches in a registry. To discover a service, a client sends out an “availability of services” request over the network. The Web Service Registry replies with details of advertised Web Services and how they should be called, in a standard format (usually WSDL). Requests and responses are normally sent using the SOAP protocol (formally known as the Simple Object Access Protocol). After successfully finding multiple matches, it chooses a service. The client then chooses a service based on its preferences. The client then downloads the service description and binds with that to invoke and use the service. The communication between client and remote Web services is generally carried out by the exchange of SOAP messages over a transport protocol, for instance HTTP.

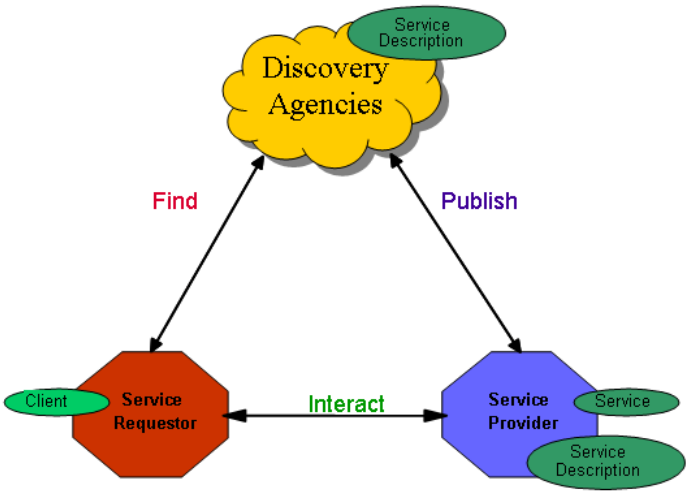


Figure 2-2: The basic Web services architecture.

2.4 **Web Services Architecture**

When analysing Web services architectures, two different aspects must be considered. According to [5], the first aspect is related to the fact that Web services are a way to expose internal operations so that they can be invoked through the Web. Such an implementation requires the system to be able to receive requests through the Web and to pass them to the underlying IT system. Such an infrastructure is referred as internal middleware for Web services. Much of the internal middleware for Web services today revolves around packing and unpacking messages exchanged between Web services and converting them into the format supported by the underlying middleware. The term internal architecture is correspondingly used to refer to the organization and structure of the internal middleware. The other facet of Web services architectures is represented by the middleware infrastructure whose purpose is to integrate different Web services. Such an infrastructure is referred as external middleware for Web services. The term external architecture is correspondingly referred to the organization and structure of the external middleware. The external architecture has three main components:

- **Centralized brokers**. These are analogous to the centralized components in conventional middleware that route messages and provide properties to the interactions (such as logging, transactional guarantees, name and directory servers, and reliability).
- **Protocol infrastructure**. This refers to the set of components that coordinate the interactions among Web services and, in particular, implement the peer-to-peer protocols whose aim is to provide middleware properties in those B2B settings where a centralized middleware platform cannot be put in place due to trust and privacy issues.
- **Service composition infrastructure**. This refers to the set of tools that support the definition and execution of composite services.

Figure 2-3 illustrates an internal and an external architecture of Web services, along with corresponding middleware support.

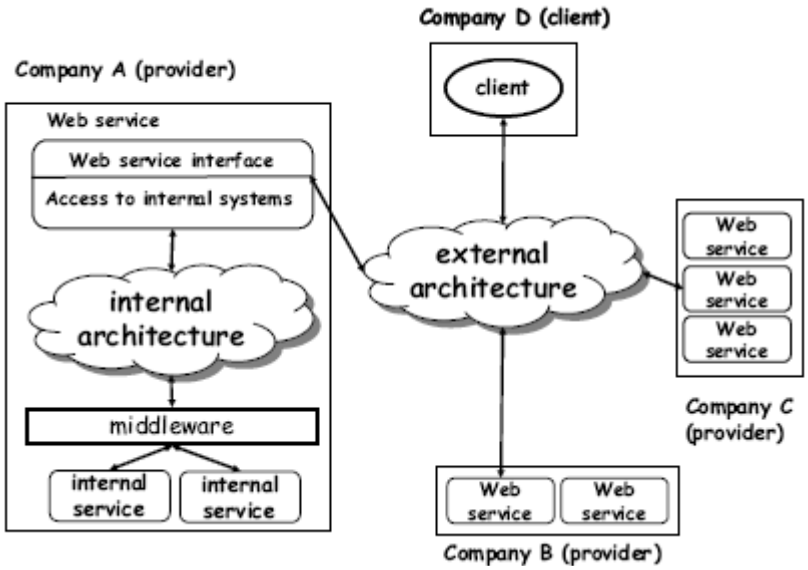


Figure 2-3: Web services require an internal and an external architecture, along with corresponding middleware support.

3. Basic Web services technology

In previous chapter, we have shown that basic Web services architecture today is based on three components: the service requester, the service provider, and the service registry, thereby closely following a client/server model with an explicit name and directory service. Such architecture illustrates quite well the basic infrastructure necessary to implement Web services: a way to communicate (SOAP), a way to describe services (WSDL), and a name and directory server (UDDI). SOAP, WSDL and UDDI are nowadays the core of Web services. The Table 3-1 lists a brief description and purpose of each. The details of each one are described in following sections.

	Purpose	**Description**
SOAP	Packaging	Simple Object Access Protocol. An XML-based protocol for encoding messages sent between a Web Service method and a client. Encodes the arguments passed to a Web Service method as well as any values returned by the method to the client.
WSDL	Description	Web Service Description Language. An XML-based protocol for describing a Web Service. A WSDL document provides the signatures of a Web Services' methods as well as other information about the data types involved in the Web Service. By referencing the WSDL, client code can use the Web Services' types and methods.
UDDI	Discovery	Universal, Description, Discovery, and Integration. An XML-based protocol for creating Web Service registries those applications can use to locate Web Service descriptions.

Table 3-1: The purpose and a brief description of SOAP, WSDL and UDDI.

3.1 **XML Fundamentals**

Table 3-1 shows that all Web services specifications are based on XML. In this section we discuss how the core Extensible Markup Language (XML) technologies on which Web services are built work, such as the XML Information Set, XML 1.0, and XML schema. For details, we refer to [7].

3.1.1 **Understanding XML**

XML is a text-based format that provides mechanisms for describing document structures using markup tags (words surrounded by '<' and '>'). As the use of XML has grown, it is now generally accepted that XML is not only useful for describing new document formats for the Web but is also suitable for describing structured data. XML is preferable to previous data formats because XML can easily represent both tabular data (such as relational data from a database or spreadsheets) and semi-structured data (such as a Web page or business document). This has led to the widespread adoption of XML as the lingua franca of information interchange. Besides being able to represent both structured and semi-structured data, XML has a number of characteristics that have caused it to be widely adopted as a data representation format. XML is extensible, platform-independent, and supports internationalization by being fully Unicode compliant. The fact that XML is a

text-based format means that when the need arises, one can read and edit XML documents using standard text-editing tools.

3.1.2 **The Anatomy of an XML Document**

Appendix A shows a sample XML document that represents a customer order for a music store. The document begins with the optional XML declaration that specifies what version of XML is being used and character encoding used by the document. This is followed by the xml-stylesheet processing instruction, which is used to bind a style sheet containing formatting instructions to the XML document for use in rendering it in a more attractive manner in user applications such as Web browsers. Processing instructions are generally used to embed application-specific information in an XML document. For instance, most applications that process the contents of the above document would ignore the xml-stylesheet processing instruction. On the other hand, applications used for displaying XML documents such as a Web browser would use the information in the processing instruction to determine where to locate the style sheet that contains special instructions for displaying the document.

3.1.3 **The Infoset**

The XML Infoset is a tree-based hierarchical representation of an XML document. An XML document's information set consists of a number of information items, which are abstract representations of the components of an XML document. There are information items representing the document, its elements, attributes, processing instructions, comments, characters, notations, namespaces, unparsed entities, unexpanded entity references, and the document type declaration. The XML Infoset is an official attempt to define what should be considered to be significant information in an XML document. For example, the infoset does not distinguish between the two forms of empty element. So “<test></test>” and “<test/>” are considered equivalent according to the XML Infoset. Similarly, the kind of quotation marks used for attributes is not considered significant.

3.1.4 **Schema Languages**

An XML schema language is used to describe the structure and content of an XML document. For instance, a schema can be used to specify a document that consists of one or more compact-disc elements which each contain a price, title, and artist element as children. During document interchange, an XML schema describes the contract between the producer and consumer of XML since it describes what constitutes a valid XML message between the two parties. Although a number of schema languages exist for XML, from DTDs to XDR, the one that currently rules the roost is the W3C XML Schema Definition Language typically abbreviated as XSD. Appendix B shows a sample schema fragment that describes the items element in the sample document in the previous section.

3.2 **Understanding SOAP**

3.2.1 **Goals of SOAP**

Simple Object Access Protocol (SOAP) is a protocol that underlies all interactions among Web services. The main point of SOAP is to provide a standardized way to encode different protocols and interaction mechanisms into XML documents that can be easily exchanged across

the Internet. In particular, it specifies the following [5]:

- A message format for one-way communication, describing how information can be packaged into an XML document.
- A set of conventions for using SOAP message to implement the Remote Procedure Call (RPC) interaction pattern, defining how clients can invoke a remote procedure by sending a SOAP message and how services can reply by sending another SOAP message back to the caller.
- A set of rules that any entity that processes a SOAP message must follow, defining in particular the XML elements that an entity should read and understand, as well as the actions these entities should take if they do not understand the content.
- A description of how a SOAP message should be transported on top of HTTP and SMTP.

3.2.2 **Structure and contents of a SOAP message**

SOAP exchanges information using *messages*. These messages are used as an envelope where the application encloses whatever information needs to be sent. Each envelope contains two parts: a header and a body. See Figure 3-1. The core of the information the sender wants to transmit to the receiver should be in the body of the message. The body is the actual message being conveyed. Any additional information necessary for intermediate processing or added value services (like security, etc.) goes into the header. Figure 3-2 illustrates a sample notification message expressed in SOAP. The message contains a SOAP header block with a local name of alertcontrol and a body element with a local name of alert. In this example an intermediary might prioritize the delivery of the message based on the priority and expiration information in the SOAP header block. The body contains the actual message payload, in this case the alert message.

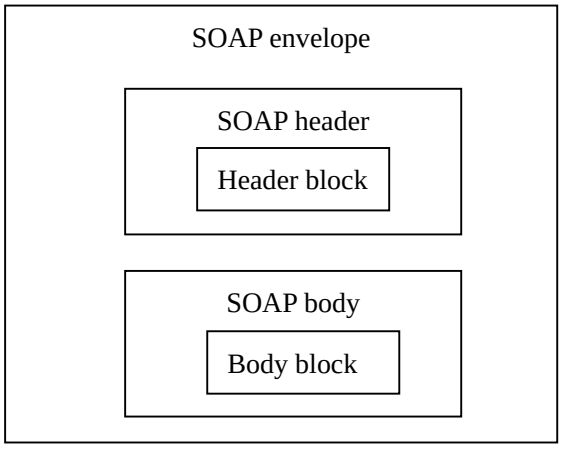


Figure 3-1: Schematic representation of a SOAP message.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
```

```
 </env:Header>
 <env:Body>
  <m:alert xmlns:m="http://example.org/alert">
    <m:msg>Pick up Mary at school at 2pm</m:msg>
  </m:alert>
 </env:Body>
</env:Envelope>
```

Figure 3-2: SOAP message containing a SOAP header block and a SOAP body

3.2.3 **Processing Model**

SOAP defines a processing model that outlines rules for processing a SOAP message as it travels from a SOAP sender to a SOAP receiver. Figure 3-3 illustrates a SOAP messaging scenario, where multiple intermediary nodes sit between the initial sender and the ultimate receiver and intercept SOAP messages.

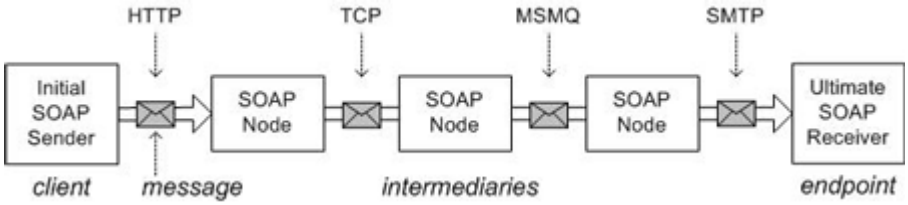


Figure 3-3: Sophisticated SOAP messaging

While processing a message, a SOAP node assumes one or more roles that influence how SOAP headers are processed. Roles are given unique names so they can be identified during processing. When a SOAP node receives a message for processing, it must first determine what roles it will assume. It may inspect the SOAP message to help make this determination. The SOAP specification defines three roles, informally called *none*, *next*, and *ultimateReceiver*:

- If a block is assigned to a none role, it means that such a block should not be processed by any node receiving the message.
- If a block is assigned to the ultimateReceiver role, that block is solely intended for the recipient of the message, not for any intermediate node.
- If a block is assigned to the next role, every node receiving the message can process that block. This is because every node receiving a message is the “next” one in the chain of nodes processing the message. The ultimateReceiver is also included in the set of next nodes.

SOAP headers target specific roles through the global actor attribute (the attribute is named role in SOAP 1.2). If the actor attribute isn't present, the header is targeted at the ultimate receiver by default. The following SOAP message illustrates how to use actor:

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
        <wsrp:path xmlns:wsrp=http://schemas.xmlsoap.org/rp
         soap:actor=http://schemas.xmlsoap.org/soap/actor/next soap:mustUnderstand="1"
         >
         ...
```

Figure 3-4: a part of SOAP message

Since the wsrp:path header is targeted at the next role and marked as mandatory (mustUnderstand="1"), the first SOAP node to receive this message is required to process it according to the header block's specification, in this case WS-Routing. If the SOAP node wasn't designed to understand a mandatory header targeted at one of its role, it is required to generate a SOAP fault, with a soap:MustUnderstand status code, and discontinue processing. The SOAP Fault element provides the faultactor child element to specify who caused the fault to happen within the message path. The value of the faultactor attribute is a URI that identifies the SOAP node that caused the fault. If a SOAP node successfully processes a header, it's required to remove the header from the message. SOAP nodes are allowed to reinsert headers, but doing so changes the contract parties—it's now between the current node and the next node the header targets. If the SOAP node happens to be the ultimate receiver, it must also process the SOAP body.

3.2.4 **Binding SOAP to a Transport Protocol**

In this section, we discuss how a SOAP message is going to be transported through the network. Typically, SOAP is associated with HTTP but it can also be used with other protocols such as SMTP (e-mail). For instance, when SOAP is used over HTTP, what is being sent is the SOAP envelope within an HTTP request. The identification of the ultimate receiver's address is not part of a SOAP message. This is resolved by including the SOAP message as part of an HTTP request or as part of an SMTP message. IN the case of HTTP, the URL of the target resource describes the receiver of the SOAP message. Similarly, in SMTP, the "to" address in the email header also describes the SOAP receiver. Figure 4 illustrates many of the SOAP HTTP binding details.

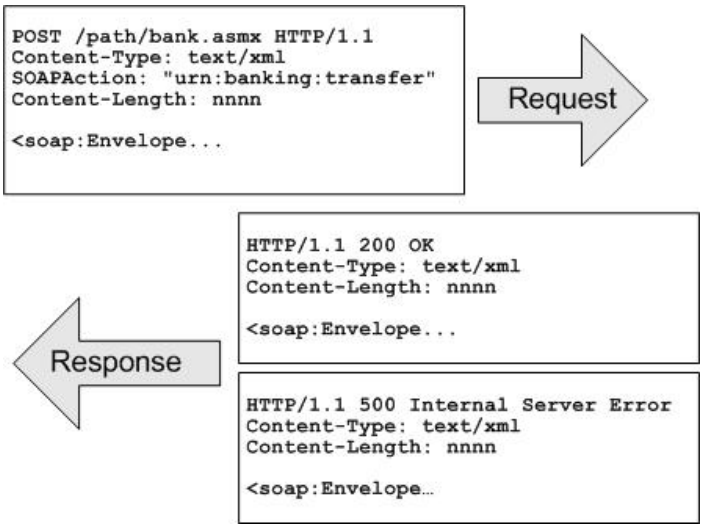


Figure 3-5: the SOAP HTTP binding

The Content-Type header for both HTTP request and response messages must be set to text/xml (application/soap+xml in SOAP 1.2). As for the request message, it must use POST for the verb and the URI should identify the SOAP processor. The SOAP specification also defines a new HTTP header called SOAPAction, which must be present in all SOAP HTTP requests (even if empty). The SOAPAction header is meant to express the intent of the message. As for the HTTP response, it should use a *200* status code if no errors occurred or *500* if the body contains a SOAP Fault.

3.3 **WSDL**

3.3.1 **Introduction**

WSDL stands for Web Services Description Language. It is an XML-based language for describing Web services: specifying the location of the service and the operations (or methods) the service exposes, and how to access them.

3.3.2 **The WSDL Document Structure**

A WSDL document is just a simple XML document. It defines a web service using the major elements: <portType>, <message>, <types>, <binding>.

WSDL Ports: The <portType> element is the most important WSDL element. It defines a web service, the operations that can be performed, and the messages that are involved. The <portType> element can be compared to a function library (or a module, or a class) in a traditional programming language.

WSDL Messages:The <message> element defines the data elements of an operation. Each message can consist of one or more parts. The parts can be compared to the parameters of a function call in a traditional programming language.

WSDL Types: The <types> element defines the data type that is used by the web service. For maximum platform neutrality, WSDL uses XML Schema syntax to define data types.

WSDL Bindings: The <binding> element defines the message format and protocol details for each port.

The main structure of a WSDL document looks like this:

```
<definitions>
<types>
    definition of types.....
</types>

<message>
    definition of a message....
</message>

<portType>
    definition of a port.....
</portType>

<binding>
    definition of a binding....
</binding>

</definitions>
```

Figure 3-6: The main structure of a WSDL document

A WSDL document can also contain other elements, like extension elements and a service element that makes it possible to group together the definitions of several web services in one single

WSDL document. For a complete syntax overview go to the Appendix C. Below, we give a simplified fraction of a WSDL document:

```
<message name="getTermRequest">
    <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
    <part name="value" type="xs:string"/>
</message>
<portType name="glossaryTerms">
  <operation name="getTerm">
      <input message="getTermRequest"/>
      <output message="getTermResponse"/>
  </operation>
</portType>
```

Figure 3-7: Simplified fraction of a WSDL document.

In this example the <portType> element defines "glossaryTerms" as the name of a port, and "getTerm" as the name of an operation. The "getTerm" operation has an input message called "getTermRequest" and an output message called "getTermResponse". The <message> elements define the parts of each message and the associated data types. Compared to traditional programming, glossaryTerms is a function library, "getTerm" is a function with "getTermRequest" as the input parameter and getTermResponse as the return parameter. In this example, the request-response type is used as the operation type. Apart from that, WSDL defines another three operation types. The table below gives a brief description of each.

Type	**Definition**
One-way	The operation can receive a message but will not return a response
Request-response	The operation can receive a request and will return a response
Solicit-response	The operation can send a request and will wait for a response
Notification	The operation can send a message but will not wait for a response

Figure 3-8: Operation types

3.3.3 **Binding WSDL to SOAP**

Below we give an example of binding WSDL document to SOAP by using the request-response operation example mentioned in the previous section.

```
<message name="getTermRequest">
    <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
    <part name="value" type="xs:string"/>
```

```
</message>
<portType name="glossaryTerms">
  <operation name="getTerm">
      <input message="getTermRequest"/>
      <output message="getTermResponse"/>
  </operation>
</portType>
<binding type="glossaryTerms" name="b1">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
  <operation>
    <soap:operation
     soapAction="http://example.com/getTerm"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
```

Figure 3-9: Binding WSDL to SOAP.

In this example the binding element has two attributes - the name attribute and the type attribute. The name attribute (you can use any name you want) defines the name of the binding, and the type attribute points to the port for the binding, in this case the "glossaryTerms" port. The soap:binding element has two attributes - the style attribute and the transport attribute. The style attribute can be "rpc" or "document". In this case we use document. The transport attribute defines the SOAP protocol to use. In this case we use HTTP. The operation element defines each operation that the port exposes.For each operation the corresponding SOAP action has to be defined. You must also specify how the input and output are encoded. In this case we use "literal".

3.4 **UDDI**

UDDI is a platform-independent framework for describing services, discovering businesses, and integrating business services by using the Internet. It is a directory for storing information about web services and a directory of web service interfaces described by WSDL. It is built into the Microsoft .Net platform. Its communications are realized via SOAP. The ultimate goal of UDDI is to streamline online transactions by enabling companies to find one another on the Web and make their systems interoperable for e-commerce. The core of UDDI revolves around the notion of business registry, which is essentially a sophisticated naming and directory service. In particular, UDDI defines data structures and APIs for publishing service descriptions in the registry and for querying the registry to look for published descriptions.

UDDI is often compared to a telephone book's white, yellow, and green pages. The project allows businesses to list themselves by name, product, location, or the Web services they offer. The UDDI

Project is a joint initiative of concerned businesses that want to advance Internet-based computing. The UDDI member companies have come together to develop an open specification and implementations of a universal business registry that is capable of integrating electronic commerce sites. This specification will be submitted to a formal standards body. The UDDI Business Registry is operated as a distributed service. It contains information about businesses and the services they offer. The information is organized as follows:

- **Business Entity**: A business entity represents information about a business. Each business entity contains a unique identifier, the business name, a short description of the business, some basic contact information, a list of categories and identifiers that describe the business, and a URL pointing to more information about the business.
- **Business Service**: Associated with the business entity is a list of business services offered by the business entity. Each business service entry contains a business description of the service, a list of categories that describe the service, and a list of pointers to references and information related to the service.
- **Specification Pointers**: Associated with each business service entry is a list of binding templates that point to specifications and other technical information about the service. For example, a binding template might point to a URL that supplies information on how to invoke the service. The specification pointers also associate the service with a service type. (A service type is defined by a tModel. Multiple businesses can offer the same type of service, as defined by the tModel. A tModel specifies information such as the tModel name, the name of the organization that published the tModel, a list of categories that describe the service type, and pointers to technical specifications for the service type such as interface definitions, message formats, message protocols, and security protocols.)

Access to and from the UDDI Business Registry is performed using SOAP. However, a service registered in the UDDI Business Registry can expose any type of service interface. Figure below illustrates how UDDI works.

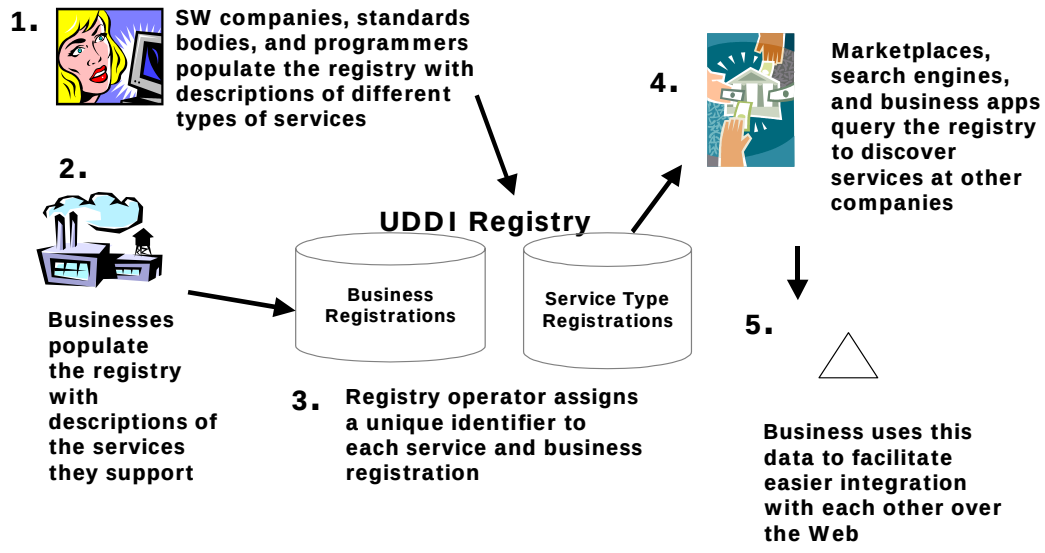


Figure 3-10: UDDI flow chart.

3.5 **What is .NET?**

.NET is the Microsoft Web services strategy to connect information, people, systems, and devices through software. It is built on the following Internet standards: HTTP, XML, SOAP and UDDI. Integrated across the Microsoft platform, .NET technology provides the ability to quickly build, deploy, manage, and use connected, security-enhanced solutions with Web services. .NET-connected solutions enable businesses to integrate their systems more rapidly and in a more agile manner and help them realize the promise of information anytime, anywhere, on any device. The Microsoft platform includes everything a business needs to develop and deploy a Web service-connected IT architecture: servers to host Web services, development tools to create them, applications to use them, and a worldwide network of more than 35,000 Microsoft Certified Partner organizations to provide any help you need. For more details about .Net technology, we refer to [8].

4. QoS for Web services

4.1 Understanding QoS for Web services

With the proliferation of Web services as a business solution to enterprise application integration, the *quality of service (QoS)* offered by Web services is becoming the utmost priority for service provider and their partners. Due to the dynamic and unpredictable nature of the Web, providing the acceptable *QoS* is really a challenging task. In addition to this, the different applications that are collaborating for Web Services interaction with different requirements will compete for network resources. The above factors will force service providers to understand and achieve Web Services *QoS*. Also, a better *QoS* for a Web service will bring competitive advantage over others by being a unique selling point for service provider. The Web Services *QoS* requirement mainly refers to the quality, both functional as well as non-functional, aspect of a Web Service. The major requirements for supporting QoS in Web services are as follows [9]:

- **Availability** represents the probability that a service is available.
- **Accessibility** represents the degree it is capable of serving a Web service request.
- Integrity is the quality aspect of how the Web service maintains the correctness of the interaction in respect to the source.
- **Performance** is the quality aspect of Web service, which is measured in terms of throughput and latency. Higher throughput and lower latency values represent good performance of a Web service. Throughput represents the number of Web service requests served at a given time period. Latency is the round-trip time between sending a request and receiving the response.
- **Reliability** represents the degree of being capable of maintaining the service and service quality.
- **Interoperability** is the quality aspect of the Web service in conformance with the rules, the law, compliance with standards, and the established service level agreement.
- **Security** is the quality aspect of the Web service of providing confidentiality and non-repudiation by authenticating the parties involved, encrypting messages, and providing access control.

[10] proposes a *QoS* stack that addresses various issues that are present in different layers of the Web Services stack, which is depicted in Figure 4-1.

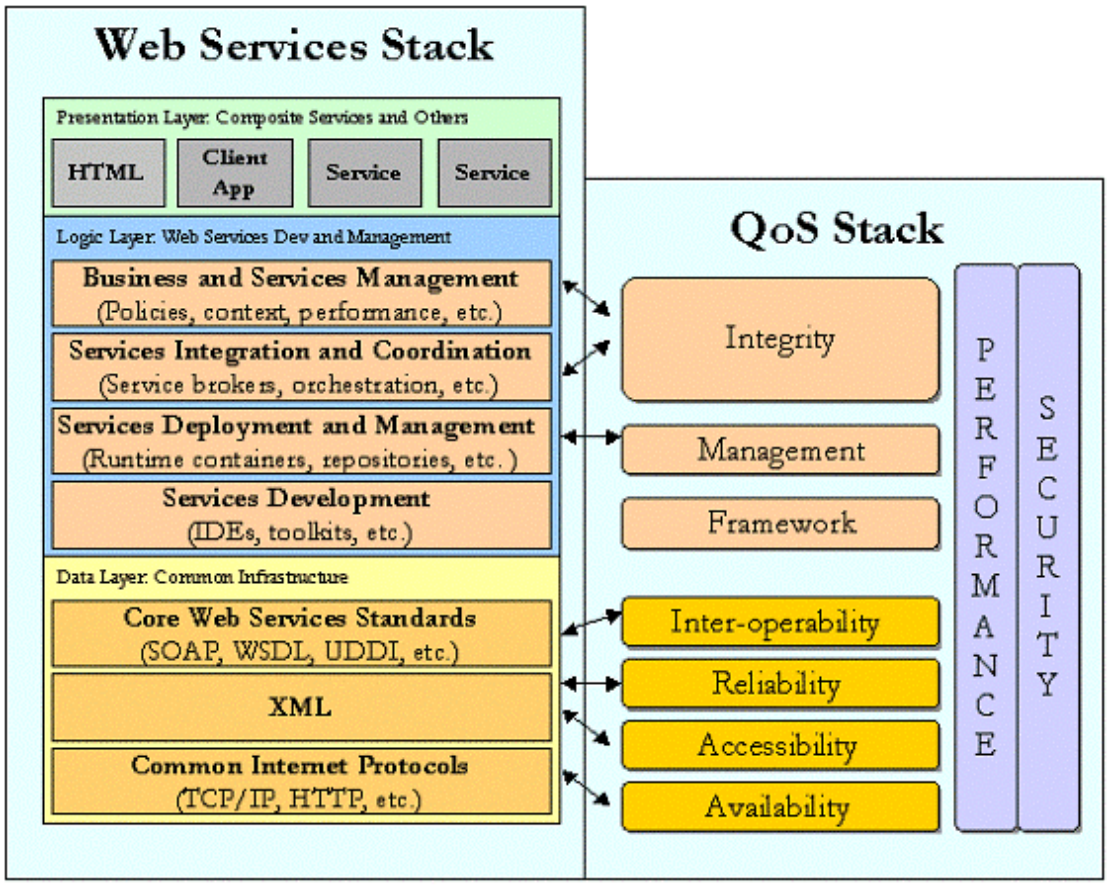


Figure 4-1: The layers of a Web Services stack

4.2 **Analysis of response-time performance of the Web services**

In this section we use the example of the travel site for purchasing a vacation package described in Section 2.2 to analyze the response-time performance of the Web services. The response time is defined as the time between sending a request and receiving the response. The QoS of the travel site may be strongly affected by the QoS of the various Web services it uses. Figure 4-2 shows a Web service flow graph (WSFG) whose nodes are either Web sites or Web services. A directed edge between nodes *a* and *b* indicates that *a* uses the services of *b*. The label on the edge (*a*, *b*), called the relative visit ratio, is the average number of times node *b* is visited per visit to node *a*. So on average, each travel booking request to the travel site generates V_a requests to the airline Web service, V_h requests to the hotel Web service, and V_c requests to the car rental Web service. Denote R_{TA} as the response time of the travel site based on the response time of the three Web services it uses, which are denoted by R_a, R_h, and R_c respectively. According to [2], it is easy to establish a lower bound on the response time R_{TA} on the response times of the three Web services it uses:

$$R_{TA} = \max\{V_a R_a, V_h R_h, V_c R_c\} \tag{4.1}$$

To see the usefulness of this equation, suppose that the response time of the airline, hotel, and car rental Web services is 0.05 sec, 0.07 sec, and 0.01 sec, respectively, and that on average, each travel site request will visit the airline Web service 4 times, the hotel Web service twice, and the

car rental service only once. So, using Equation (5.1), we have

$$R_{TA} = \max\{0.2, 0.14, 0.01\} = 0.2 \text{ sec} \tag{4.2}$$

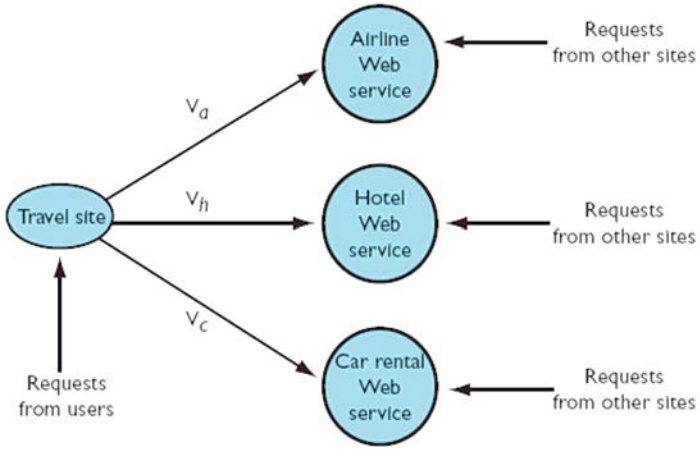


Figure 4-2. Web service flow graph. Arrows link the travel site to other Web services.The labels on the links indicate the average number of times a Web service is invoked per request to the travel site.

Equation (4.2) says that in order for the travel site to decrease the lower bound on its response time, it would need to use a better airline Web service, because this is the Web service that limits the minimum response time of the travel site. Alternatively, the travel site could try to reduce the number of times it has to invoke the airline Web service per transaction.

Assume R_a, R_h, and R_c are independent variables and all exponentially distributed with the same mean S. Based on (4.1), the probability that the R_{TA} is t seconds or less can be expressed as follows:

$$p(R_{TA} \le t) = p(R_a \le t)p(R_h \le t)p(R_c \le t). \tag{4.3}$$

This is equal to

$$p(R_{TA} \le t) = \left(1 - e^{-St}\right)^3. \tag{4.4}$$

In real life, to guarantee the response time performance of Web services, the travel site may send requests to Web services concurrently. These Web services have the same functionality, but the quickest reply is accepted. Another benefit to do this is if there are some Web services failed, then the request can still be handled without delay. Also there is the situation that in order to invoke a certain Web service, part information need to be achieved first from other Web services.

Consider the situation that you are driving on the highway. The destination is still far away, but you are exhausted and the gasoline is almost run out. In order to find out the information about hotels and gas stations nearby, you send the request via your PDA to the service provider to achieve all available hotels and gas stations within 5 km. As the first step, the service provider has to find out where you are by using the location Web service. After that, the service provider uses the gas station Web service and the hotel Web service to get a list of hotels and gas stations you asked for. Assume there are two location Web service providers and two hotel web service providers. The diagram is shown in Figure 4-3. Denote R, R_{L1}, R_{L2}, R_G, R_{H1}, and R_{H2} as the response time of the service provider, location Web service 1, location Web service 2, gas station Web service, hotel Web service 1, and hotel Web service 2 respectively. To simplify the problem

we assume:

- R_{L1}, R_{L2}, R_G, R_{H1}, and R_{H2} are independent and are all exponentially distributed with the same mean S.
- Each request to the service provider generates only 1 request to all Web services.

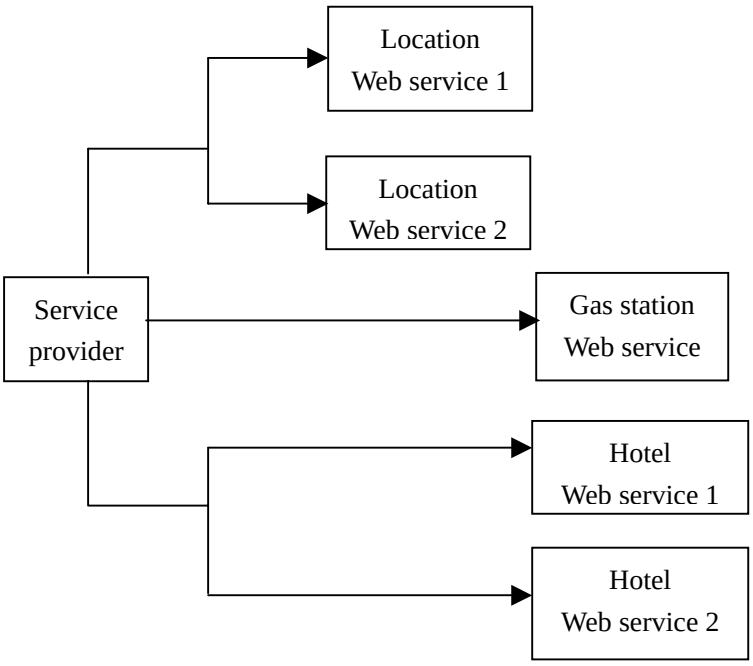


Figure 4-3: A Web service example for finding available hotels and gas stations

Then it is easy to verify that

$$R = \min(R_{L1}, R_{L2}) + \max(R_G, \min(R_{H1}, R_{H2})) \tag{4.5}$$

Based on (4.5), the probability that R is t seconds or less can be derived. At first, we denote:

$$X = \min(R_{L1}, R_{L2}), \tag{4.6}$$

$$Y = \min(R_{H1}, R_{H2}), \tag{4.7}$$

and

$$Z = \max(R_G, Y). \tag{4.8}$$

Then we have,

$$P(R \le t) = P(X + Z \le t). \tag{4.9}$$

This is equal to

$$P(R \le t) = \int_0^t \int_0^{t-x} f_X(x) f_Z(z) dz dx, \tag{4.10}$$

where $f_X(x)$ and $f_Z(z)$ represent the distribution function of X and Z respectively.

According to the first assumption, it is easy to verify that X and Y are both exponentially distributed with the mean $2S$. The distribution of Z can be calculated via the follows steps. Because

$$P(Z \le x) = P(R_G \le x)P(Y \le x), \tag{4.11}$$

we get

$$P(Z \le x) = 1 - e^{-Sx} - e^{-2Sx} + e^{-3Sx}. \tag{4.12}$$

According to (4,12), we get the distribution of Z as follows:

$$f_Z(x) = S\,e^{-Sx} + 2S\,e^{-2Sx} - 3S\,e^{-3Sx}. \tag{4.13}$$

Combining (4.10) and (4.13), we get the probability that R is t or less as follows.

$$P(R \le t) = e^{-3tS}[-2e^{2tS} + 3e^{tS} - 2tS\,e^{tS} - 2] + 1, \tag{4.14}$$

4.3 **Charging Mechanisms**

Before selling a Web Service, Web services publishers need to decide how they are going to charge for it. [11] lists some simple charging structures outlined below:

Charge per call (Prepaid)	A customer buys a set number of calls to one of Web Services. Every time a Web service is called by this customer, the call is registered. When the customer runs out of calls, s/he is automatically notified and asked to buy more.
Charge per month (Subscription)	A customer pays for unlimited use over a period of time. S/he pays to have access to a Web Service for a particular period of time: one month, three months, six months, or one year. Once that period ends, the customer is notified and asked to confirm that s/he wants another month's usage or, depending on our arrangement, automatically charge their account for another period of time.
One-off charge (Prepaid)	The customer makes a one-off payment for unlimited use of Web Service for the lifetime of that Web Service. The lifetime might only be a few months: for instance, with the case of a news feed for a specific Olympic games.
No charge (Freeware)	The Web Service is free for a specific period. This could be for the lifetime of the Web Service or for a shorter trial period.

Web services publishers don't have to use only one of the above charging methods for Web service. They could combine any of the above methods to make alternative payment schemes. For instance, they could start with the 'No charge' method for a trial period before using the 'Charge per call' method.

4.4 **A model for minimizing the cost**

In Section 4.2, we discussed the response time without taking the cost into account. But sometimes the service provider has to pay the cost for some Web services. Normally shorter response time or faster server will be more expensive. In this section we develop a model for minimizing the cost while considering an acceptable response time. We use the example described in Section 4.2 about finding available hotels and gas stations via Web services. To simplify the problem, we assume that there is just one location Web service and hotel Web services are not considered. See Figure 4-4.

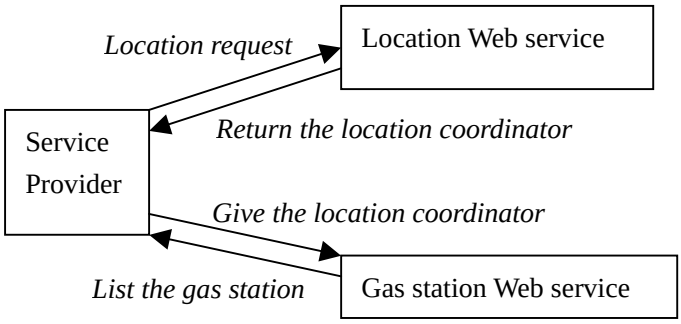


Figure 4-4: A Web service example for finding available gas stations

The variables used in this example are defined as follows,

R: the average time between the service provider receives the request from the user and the service provider gets the lists of the gas station.

R_L: the mean Response time of the location Web service.

R_P: the mean Response time of the gas station Web service.

C: the minimum cost

C_L: Cost of visiting the location Web service every time.

C_P: Cost of visiting the Gas station Web service every time.

To simplify the problem, in this example we make the following assumptions:

- Both Web services use “charging per call” method.
- For each complete request, the service provider generates one request on average to both the location Web service and the Gas station Web service.
- The time interval between receiving the location coordinator from the location Web service and forwarding it to the gas station Web service can be ignored.
- *R* should not exceed *r* seconds to get a satisfying QoS.

In general, shorter response time or faster service will be more expensive. Therefore, we assume

$$C_L = \frac{a}{R_L}, \quad \text{where} \quad a \geq 0, \tag{4.15}$$

and

$$C_P = \frac{b}{R_P}, \quad \text{where} \quad b \geq 0. \tag{4.16}$$

Then we have the following expressions for minimizing the cost while taking the acceptable response time into account.

$$\min (C_L + C_P) \tag{4.17}$$

under the conditions

$$R_L + R_P \leq r$$
$$R_L \geq 0, R_P \geq 0.$$

Substituting (4.15) and (4.16) into (4.17), we have

$$\min \left(\frac{a}{R_L} + \frac{b}{R_P}\right) \tag{4.18}$$

where

$$R_L + R_P \leq r$$
$$R_L \geq 0, R_P \geq 0.$$

From Figure 4-5, it is easy to know that the solution of (4.18) must lies on the linear line

$$R_L + R_P = r. \tag{4.19}$$

It can be proved by apagoge. Suppose that under that linear line, there exists a point (R'_L, R'_P) that can get the minimum cost. Then a point (R_L, R_P) can be found on the linear line which holds that $R'_L < R_L$ and $R'_P < R_P$. See figure below. This follows that the cost corresponding to (R_L, R_P) is lower than the minimum cost. This is infeasible. Therefore the best solution must be found on that linear line.

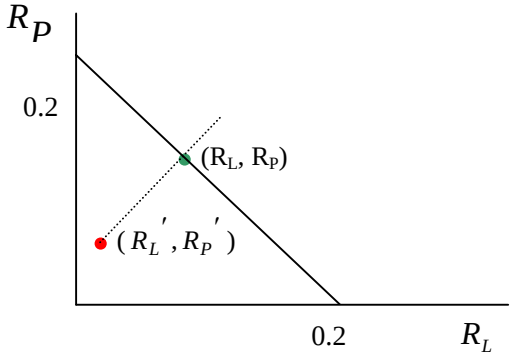


Figure 4-5: Condition ar

Hence, (4.18) can be converted to

$$\min \ (\frac{a}{R_L} + \frac{b}{r - R_L}) \tag{4.20}$$

By setting the differential of the equation expressed in (4,20) to zero, we get the following conclusion: when

$$R_L = \frac{r}{1 + \sqrt{b/a}}, \tag{4.21}$$

and

$$R_P = \frac{r}{1 + \sqrt{a/b}} \tag{4.22}$$

the minimum cost is achieved. That is equal to

$$C = \frac{a + b + 2\sqrt{ab}}{r}. \tag{4.23}$$

This minimum cost is based on “charging per call” mechanism. This mechanism is suitable if the times that the service provider uses a Web service are limited. But if the service provider must invoke a Web service frequently, is it not better to take a subscription? Consider the month June, July, and Augustus. In these months, the amount of people on the way increased dramatically because most people are on holiday. This may cause the increasing of the number of requests mentioned in our example. In this case, which decision should the service provider make, “charge per call” or “charge per month”? At first, let us assume:

- The number of requests to use the Web service illustrated in Figure 4-4 is distributed according to a Poisson distribution with the parameter μ.
- The cost of subscription is *M* euro per month.
- The QoS is guaranteed by the subscription.
- The cost for “charge per call’ is *C* euro per time. *C* is expressed in (4.23)

It is easy to retrieve that if the number of times the Web service invoked is lower than *M/C*, then using “charge per call” mechanism is more economical. That implies if

$$P(x \le \frac{M}{C}) \ge 0.5, \tag{4.24}$$

then it is better to take “charge per call”.

For instance, suppose that $\mu = 15$, using the cumulative Poisson distribution table (4.24) follows the result:

$$\frac{M}{C} \ge 14. \tag{4.25}$$

This result implies that if $M \ge 14C$, for the case that $\mu = 15$, “charge per call” is our preference. Otherwise it is better to chose “charge per month”.

5. Summary

In this paper we first describe what Web services are, how they are built. We have followed the W3C definition of a Web service, described as "*a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols*". Web services can be characterized by an internal architecture, defining its connection with the local information systems, and an external architecture, defining how Web services discover and interact with each other.

Secondly, we discussed what technologies are required for Web services. Then we described each required Web service technology in detail. In Web services model, Web service providers use the Web Services Description Language (WSDL) to describe the services they provide and how to invoke them. The service providers then register their services in a public service registry using universal description, discovery, and integration (UDDI). Application programs discover services in the registry and obtain a URL for the WSDL file that describes the service. Then the applications can invoke the services using the XML-based simple object access protocol (SOAP).

Finally, we discussed the QoS issues of Web services and charging structures. The Web services QoS requirement mainly refers to the quality, both functional as well as non-functional, aspect of a Web service. This includes *performance*, *reliability*, *integrity*, *accessibility*, *availability*, *interoperability*, and *security*. In our paper, we focused on the response time performance of a Web service. We built a model to calculate the probability that the response time R_{TA} is t seconds or less. Apart from that, we discussed various charging mechanisms: *charge per call*, *charge per month*, *on-off charge*, and *no charge*. After that, we built another two models for finding out how to satisfy the customer at lowest price and when the "charge per call" mechanism is better than "charge per month" respectively.

Bibliography

[1] E.M. Maximilien and M.P. Singh, “A Framework and Ontology for Dynamic Web Services Selection,” IEEE Internet Computing, vol. 8, no. 5, 2004, pp. 84–93.

[2] D.A. Menascé, “QoS Issues in Web Services,” IEEE Internet Computing, vol. 6, no. 6, 2002, pp. 72–75.

[3] W3C. Web Services Architecture Requirements, http://www.w3.org/TR/wsa-reqs, Oct. 2002.

[4] F. Curbera et al., “Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI,” *IEEE Internet Computing*, vol. 6, no. 2, 2002, pp. 86–93.

[5] G. Alonso, F. Casati, H. Kuno and V. Machiraju, “Web services, concept, Architectures and Applications”, 2004.

[6] D. McDonald, University of Strathclyde, “Web Services Technologies Report for the JISC Technology Watch Service”, http://www.jisc.ac.uk/uploaded_documents/tsw_03-04.doc.

[7] http://msdn.microsoft.com/webservices/understanding/xmlfundamentals/default.aspx

[8] http://www.w3schools.com/ngws/default.asp.

[9] http://www-106.ibm.com/developerworks/library/ws-quality.html?n-ws-1172.

[10] http://www.developer.com/services/article.php/2027911

[11] http://www.webservicesarchitect.com/content/articles/clark03.asp

Appendix A. Sample XML document

This sample is a customer order for a music store

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<?xml-stylesheet href="orders.xsl"?>

<order id="ord123456">
   <customer id="cust0921">
      <first-name>Dare</first-name>
      <last-name>Obasanjo</last-name>
      <address>
         <street>One Microsoft Way</street>
         <city>Redmond</city>
         <state>WA</state>
         <zip>98052</zip>
      </address>
   </customer>
   <items>
      <compact-disc>
         <price>17.55</price>
          <artist>Baby D</artist>
          <title>Lil Chopper Toy</title>
      </compact-disc>
   </items>

   <!--    Always go the extra mile for the customer -->
   <special-instructions xmlns:html="http://www.w3.org/1999/xhtml/">
      <html:p>If customer is not available at the address then attempt leave package at
         one of the following locations listed in order of which should be attempted first
      <html:ol>
         <html:li>Next Door</html:li>
         <html:li>Front Desk</html:li>
         <html:li>On Doorstep</html:li>
      </html:ol>
      <html:b>Note</html:b> Remember to leave a note detailing where to pick up the package.
      </html:p>
   </special-instructions>
</order>
```

Appendix B. Sample XML schema fragment

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:element name="items">
      <xs:complexType>
           <xs:sequence>
                <xs:element ref="compact-disc" minOccurs="0" maxOccurs="unbounded" />
           </xs:sequence>
      </xs:complexType>
 </xs:element>

 <xs:element name="compact-disc">
      <xs:complexType>
           <xs:sequence>
                <xs:element name="price" type="xs:decimal" />
                <xs:element name="artist" type="xs:string" />
                <xs:element name="title" type="xs:string" />
           </xs:sequence>
      </xs:complexType>
 </xs:element>

</xs:schema>
```

Appendix C. A full WSDL 1.2 syntax in the W3C Working Draft

```
<wsdl:definitions name="nmtoken"? targetNamespace="uri">
    <import namespace="uri" location="uri"/> *
    <wsdl:documentation .... /> ?
    <wsdl:types> ?
        <wsdl:documentation .... /> ?
        <xsd:schema .... /> *
    </wsdl:types>
    <wsdl:message name="ncname"> *
        <wsdl:documentation .... /> ?
        <part name="ncname" element="qname"? type="qname"?/> *
    </wsdl:message>
    <wsdl:portType name="ncname"> *
        <wsdl:documentation .... /> ?
        <wsdl:operation name="ncname"> *
            <wsdl:documentation .... /> ?
            <wsdl:input message="qname"> ?
                <wsdl:documentation .... /> ?
            </wsdl:input>
            <wsdl:output message="qname"> ?
                <wsdl:documentation .... /> ?
            </wsdl:output>
            <wsdl:fault name="ncname" message="qname"> *
                <wsdl:documentation .... /> ?
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:serviceType name="ncname"> *
        <wsdl:portType name="qname"/> +
    </wsdl:serviceType>
    <wsdl:binding name="ncname" type="qname"> *
        <wsdl:documentation .... /> ?
        <-- binding details --> *
        <wsdl:operation name="ncname"> *
            <wsdl:documentation .... /> ?
            <-- binding details --> *
            <wsdl:input> ?
                <wsdl:documentation .... /> ?
                <-- binding details -->
            </wsdl:input>
            <wsdl:output> ?
                <wsdl:documentation .... /> ?
                <-- binding details --> *
```

```
                    </wsdl:output>
                    <wsdl:fault name="ncname"> *
                        <wsdl:documentation .... /> ?
                        <-- binding details --> *
                    </wsdl:fault>
                </wsdl:operation>
            </wsdl:binding>
            <wsdl:service name="ncname" serviceType="qname"> *
                <wsdl:documentation .... /> ?
                <wsdl:port name="ncname" binding="qname"> *
                    <wsdl:documentation .... /> ?
                    <-- address details -->
                </wsdl:port>
            </wsdl:service>
</wsdl:definitions>
```