Department of Electronic Engineering

PROJECT REPORT

BScIT-2001/02-CS/HWC-02

Development of Web Services using SOAP

Student Name: Wong Tak Hei
Student ID: 50194690
Supervisor: Dr Chun, Andy H W
Assessor: Dr Lau, Ricky W H

Bachelor of Science (Honours) in Information Technology
Abstract

Currently, most of the information on the Internet is presented in a form which is readable by human, but hard to be interpreted by applications. There exist technologies like the Java RMI and Microsoft DCOM which enable distributed computing over networks. These technologies usually require both sides be using the same language and running on the same platform. While such approach would be nice within a local network, no one can control those factors over the Internet. Enterprises needing to interact with each other would require undergoing much negotiation and integration of technology.

Web Services is a new way for distributed computing over the Internet. Web Services enable business processes to describe themselves and allow others to locate and invoke them. By using standard protocols like SOAP, WSDL and UDDI (which are all based on XML), enterprises can then be quicker and more efficient to develop applications that are shared over the Web.

In this project, the Web Services frameworks and its relevant technologies would be studied. An implementation of web applications utilizing Web Services would then be built.
Contents

INTRODUCTION 1

CHAPTER 1: BACKGROUND OF WEB SERVICES 3
   HISTORY OF DISTRIBUTED COMPUTING 3
   WEB SERVICES ARCHITECTURE 4
   CURRENT STATUS 5

CHAPTER 2: WEB SERVICES PROTOCOLS 7
   XML BASICS 7
   SOAP BASICS 9
   SOAP MESSAGES 11
   WSDL BASICS 12
   WSDL EXAMPLES 13
   UDDI BASICS 15

CHAPTER 3: DESIGN AND IMPLEMENTATION 19
   METHODOLOGY 19
   ENVIRONMENT SETTINGS AND TOOLS USED 19
   SYSTEM STRUCTURE 21
   DEVELOPMENT 25
   MESSAGES FORMAT 27
   SCREENSHOTS 29

CHAPTER 4: ANALYSIS 32
   PERFORMANCE 32
   DEVELOPMENT EFFICIENCY 32
   INTEROPERABILITY 32
   DIFFICULTIES 33
   IMPROVEMENTS 34

CONCLUSION 35

REFERENCES 36

APPENDIX: SELECTED SOURCE CODES 37
   CURRENCY EXCHANGE RATE SERVICES 37
   HOTEL RESERVATION SERVICES 40
   CLIENT-SIDE JSPS 45
Introduction

Web Services is a new kind of technology that facilitates the development of distributed computing over the Internet. Currently, most information provided on the Internet is intended to be read by human on the screen. While this is probably not an issue for consumer end users, corporate systems would usually be highly desirable be able to let the computer automatically retrieve and process data. Web Services provided an interface for systems to do just that. It provides information and services in a structuralized form to aid process automation.

Another main characteristic of web services is that it is built on open standard protocols which are all XML based. As a result, such applications would have a high degree of interoperability with minimal system integration. This is important as it is impossible to control the system environment of different service providers all over the world.

Also, web services are self descriptive. By using WSDL (Web Services Definition Language) files, a service can define the interface and other important details in order for others to consume it.

Moreover, web services can be dynamically discover and bind by utilizing a registry called UDDI (Universal Description, Discovery and Integration). The registry can be either global (public) or private. Global registry has the advantage of being able to access publicly and globally, much like the global DNS (Domain Name System) registry. Private registry, on the other hand, can have a more restrictive control over the service published and thus the service quality and integrity can be more guaranteed. Besides of the wide variety of information stored on the registry, it also has a set of UDDI API (Application Program Interface) that allows applications to programmatically access the database.
In the implementation, several web services were deployed. A web site were setup and act as a service consumer and an interface for the customer. Since this implementation was primarily used to study the techniques for building web services, the business logic of the applications will not be too realistic.

Chapter 1 briefs the evolution of distributed computing and the reason for the emergence of web services. It gives an overview for the architecture of web services. It also describes the current development status for the Web Services and its related technologies.

Chapter 2 introduces the fundamentals protocols and languages that build up Web Services: XML, SOAP (Simple Object Access Protocol), WSDL and UDDI. SOAP is a messaging protocol for transmission of information, while WSDL and UDDI are for interface definition and dynamic binding respectively.

Chapter 3 describes the actual implementation of this project. It talks about the methodology and the reason for the environment settings. An in-depth structure of the system and messaging format is included. Finally there are screenshots of the application to show the data flow for this system.

Chapter 4 would analysis the implementation system from various perspectives.
Background of Web Services

History of Distributed Computing

Long ago in the 50s when computer were first introduced, there was hardly any kind of communication between computers. As network technologies evolve and the increase popularity of PC, there was more need for application-to-application communication between different machines. By the early 90s, there were models such as COBRA/IIOP from OMG (Object Management Group) and DCOM from Microsoft that enabled computer linked by networks to communicate efficiently using object frameworks. Later, Sun Java’s RMI (Remote Method Invocation) were also introduced to perform distributed computing on Java platform.

These protocols usually have an efficient however complex structure. Usually, programmer did not need to get their hands dirty by digging into the low-level communication issues. Therefore, using these protocols to perform distributed computing would be a good choice.

Due to their complexity and incompatibility, one of the issues becomes obvious: system utilizing different technologies simply cannot (easily) talk with each other. Although such problems may not be too serious if companies are able to run the same platform on their machines, it would be an issue if the scope is across companies and countries over the Internet. There were attempts to implement different protocols to communicate with each other, but all of them had little success.

As the Internet becomes more and more popular, people are start to think of ways to create web applications that can be communicate with high level
of interoperability and automation. This is why the concept of Web Services had emerged.

**Web Services Architecture**

The architecture of web services can be divided into four parts:

- Transportation
- Invocation (Messaging)
- Description (Meaning)
- Discovery

This section briefs the four parts and how they work together from a service provider’s perspective. Details of the protocols are covered in Chapter 2.

**Transportation**

Web services do not use a specific protocol for transmission. The most common way is to use HTTP protocol. However, all other protocols (e.g. SMTP, FTP) can do just fine.

**Invocation**

Invocation is focusing on how to establish connection and pass messages between endpoints. SOAP is the key protocol for this part. As a communication protocol, it defines the necessary syntax and structure to communicate with each other. Application developed in different platform can be exposed as web services by installing and configuring SOAP components on their servers. When the SOAP components received SOAP request, it can then redirect the request to the native programs for process. The result (if any) would then been encapsulated in a SOAP messages and send back to the requestor.

**Description**

Description contains information regarding on what it can receive and return, what functions it provides, calling syntax, etc. WSDL is responsible for this part. After a web services was deployed, WSDL file is generated for the specific services. By referring to the WSDL files, service requestor can
understand the way to bind to it and create appropriate SOAP messages to initiate communication.

**Discovery**
Discovery concerns with how the world can locate and call a web service. The UDDI specification allows service providers a way to advertise its web services in a directory. It contains information ranging from company name and contacts, geographic location and most importantly, their web services provided (in WSDL format).

**Working Together**
Following is a diagram showing that how the four parts above combine the form the web service framework.

**Current Status**
Web Services is a relatively new technology, and most of the protocols are still under development. Up to the moment when writing this report, SOAP is in its version 1.2. The next version of SOAP, which is called XML Protocol (XMLP), is under development.

Besides the core protocols, there are also a lot of extension for web services under study. One of the most important issues is the Web Services Security. The WS-Security specification, jointly announced by IBM, Microsoft and Verisign in February 2002, is target to provide a roadmap for secure messaging and authentication.
While the hype of web services revolutionize the Internet had been there for some time, the actual implementation of web services is still very limited. Most of them are trivial user login or simple stock-quote and weather enquiry. However, with leading developer like Microsoft (.NET Platform) and Sun (Sun ONE) targeting their next-generation on web services, it is promising that it will take a further step of becoming the mainstream technology in the near future.
Web Services Protocols

XML Basics

Before going into the details of the web services protocols, XML is first introduced. XML is the basis for the whole web services framework. Therefore it is very important to understand what is XML before continue with the web services protocols.

XML is a subset of SGML (Standard Generalized Markup Language) which is targeted for use over the Internet. It is used to represent and exchange structured data using self-defined tags. XML is designed for easy creation and processing. It should also be self-explanatory and human-readable. Following is a very simple XML document:

```xml
<harddisk>
  <name>IBM Fireball 500</name>
  <size unit="GB">40</size>
  <mode>SCSI</mode>
</harddisk>
```

The data between open (e.g. `<mode>`) and close tag (`</mode>`) are called value (e.g. “40”) while the data inside the open tag are called attributes (e.g. `unit="GB"`). It have a hierarchical structure. In the example above, element name is a child of harddisk and harddisk is the parent of name. harddisk is also the root element for the example.

Since all elements names are self-defined, it is common that name conflict will occur when they are using the same name to represent different types.

```xml
<paper>
  <title>Some Research</title>
  <author>Jane Doe</author>
</paper>
```
If these two XML are used together, it will cause an naming conflict. To avoid the problem, prefix and namespaces attribute are used.

```
<w:paper xmlns:w="http://www.woodycompany.com/papers">
  <w:size>A4</w:size>
  <w:color>white</w:color>
  <w:weight>80</w:weight>
</w:paper>
```

For the example above, the prefix is \textit{w} and the namespace value is the URI (Uniform Resource Identifier). All namespace attributes have the syntax \texttt{xmlns:prefix="namespace"}. Also, a prefix can be omitted.

The value of a namespace is used to uniquely identify it and the XML specification states that the value should be an URI. Note that the URI is merely to define a value uniquely, what the URI are actually pointing to is not the concern for the XML file. However, it is often that the namespace value do point to a page that is relevant to the document.

Sometimes it may need to set some rules on XML document to define what kind of structure is legal or illegal. XML Schema Definition (XSD) is a language for performing the task. Following is a simple XML Schema that defines the paper element above:

```
<xs:schema xmlns:xsd="http://www.woodycompany.com/paper/schema">
  <xs:element name="paper">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="size" type="xsd:string"/>
        <xs:element name="color" type="xsd:string"/>
        <xs:element name="weight" type="xsd:int"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The above schema had type attribute equals to values like \texttt{xsd:string}. This is because XML schema had a lot of built-in data types. Some of them are string, decimal, integer and boolean.
**SOAP Basics**

Simple Object Access Protocol (SOAP) is the most important protocol of the web services framework. It is a messaging protocol that allows structured data to be transmitted in XML documents in a decentralized, distributed environment and to define the structure to invoke web services. Since the introduction of SOAP by Microsoft, most of the major vendors like IBM, Sun and Oracle had adopted it. Since SOAP is independent of implementing platforms and language, it is especially attractive to the integration of e-business among different companies.

The SOAP 1.2 Working Draft submitted to the World Wide Web Consortium (W3C) specifies the following:

- Syntax for the XML documents (i.e. SOAP messages)
- Messages exchanging model
- Rules for encoding various data types (i.e. SOAP encoding)
- Guideline for transmission of SOAP messages over HTTP
- Structure for performing both Remote Procedure Calls (RPC) and messaging

SOAP is important because it does the actual access to the services. Other protocols like WSDL and UDDI of course are useful in enhancing the service, but they are optional. SOAP is the core protocol that enables the communication to take place.

As the name implies, SOAP is designed to be a simple protocol. Therefore, the restriction on issues like syntax and transportation are set to a minimum. This makes the protocol more flexible, customizable and extensible.

SOAP is a messaging protocol, and not a transmission protocol. SOAP itself does not include a specific way to transfer between endpoints. Instead, it can use any of the existing protocols for transmission. The most common means for transport SOAP messages today is the use of HTTP protocol.
Other protocols like SMTP (email) and FTP can also work, as long as the receiving side has an agreement on how to process the received data.

Besides traditional client-server model, SOAP messages can also be optionally exchanged in a chain of location in which it will pass through several SOAP nodes before arriving at the ultimate destination. These nodes and called SOAP intermediaries. They can provide value-added function or simply to record some events before passing the message into the next node. These activities are application-specific.

From the “O” in SOAP, one may think that it is bound to some specific object model. This is not the case. The fact is that any kind of language, whether it is object-oriented or not, can be encapsulated into a SOAP message.

SOAP encoding defines a set of rules to encode various data types into SOAP messages. For example, it had specification on storing common types like string, integer, arrays, etc. It also specifies the format to encode the more complex data types like self-defined structure. By standardizing the encoding standards, different platform can easily convert the SOAP messages back to its native presentation for further manipulation.

There exist two type of service access method, namely RPC and messaging. For RPC type, the side requesting a service will send appropriate parameters and the function name to call in a SOAP message to the server. When the server accepts the request, it will process and return some information back to the sender in the same run. Messaging, on the other hand, will accept all incoming request (messages) and process them in a later time. These two can be analogous to placing a phone call and sending a letter respectively.
SOAP Messages

After describing the functionality of SOAP, this session examines the structure of it. The following diagram shows how a SOAP messages is composed.

![SOAP Envelope Diagram]

**Envelope**

It is the outermost structure that enclosed all other elements of SOAP messages. The envelope namespace indicates the SOAP version used. Currently the only valid namespace for SOAP version is the URI listed in the message below.

```xml
<soap:Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
  ...
  </soap:Body>
</soap:Envelope>
```

**Header**

It is used to carry application-specific extensions that is intended to be read by all SOAP nodes within the message path.

```xml
<soap:Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <h:booking xmlns="http://www.booking.com/ws"
      soap:mustUnderstand="true">
      <h:priority>TOP</h:priority>
    </h:booking>
  </soap:Header>
  <soap:Body>
  ...
  </soap:Body>
</soap:Envelope>
```
In the example above, it specified the booking request is of top priority. So, all SOAP nodes that receive this request shall process it first.

Also, there is an attribute called “mustUnderstand”. If it is set to false, any node which don’t understand the header information will simply ignore it. If it is true, all nodes must understand it or it will generate a fault.

A header is optional. However if it is presented, it must be the first element of the envelope.

**Body**

It contain information that are targeted to the final destination of the SOAP receiver. A Body is compulsory in SOAP messages.

Consider the definition of a C++ struct:

```c++
struct member
{
    string username
    string password
    int auth_code
};
```

A SOAP messages that call a remote login function by passing the struct above would look like this:

```xml
<soap:Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <m:Login xmlns:m="http://www.example.com/login" soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
            <member xsi:type="n:member">
                <username xsi:type="xsd:string">John Doe</username>
                <password xsi:type="xsd:string">abc123</password>
                <auth_code xsi:type="xsd:int">941239</auth_code>
            </member>
        </m:Login>
    </soap:Body>
</soap:Envelope>
```

**WSDL Basics**

WSDL is used to describe the interfaces and bindings of web services. Web services that use WSDL to publish their interface can be called by any web services client programmatically without knowing about the implementation details of them. WSDL is also based on XML.
A WSDL document defines the following:

- Types
- Messages
- Operations
- Port Types
- Bindings
- Ports
- Services

The following section will use WSDL examples to explain the meaning of the above elements.

**WSDL Examples**

**Types**
It describes the data types or structures used during communication. The following example describes a data type called “Player”, which have two elements: name (string) and number (integer).

```xml
<types>
  <xsd:schema ...>
  <xsd:element name="Player">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="number" type="xsd:int"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  </xsd:schema>
</types>
```

**Messages**
It defines the message format like parameters and outputs. The following example defines the request/response message for a function called “GetPlayerTeam”, which take an object of data type “Player” (defined above) and return a string. All messages have “part” elements that refer to data types defined in “types”.

```xml
</xsd:element>
</xsd:schema>
</types>
```
Operation & Port types
The above example defines two messages, so how can it be called? The operation defines the name and input/output messages of a function. Port types act like an envelope to group all functions provided by the service.

Bindings
It defines how a function can bind to a specific protocol (e.g. HTTP). It is similar to the port types above except that each operation is refer to a specific protocol. The example bellows show the HTTP binding information.

Ports & Services
A port specifies the binding address specific to a protocol. This is the actual endpoint of which communication take place. A service is an envelope to group all of the ports.
Putting Them Together

By publishing WSDL files that contain the seven components above, any services consumer can refer to it and build suitable application to invoke the services.

It is common that WSDL is separated into two files. One file contains the ports and services components (provider’s WSDL) while all others go to the other file (interface WSDL). This structure is useful when some bodies want to publish some service type definition without the actual service binding address. All providers which implement the services will publish their WSDL file that import the interface WSDL file and defines the binding location in provider’s WSDL.

Although we have said that web services can be self-describing using WSDL, fully automation by computer is still far from realistic. This is because, after all, those descriptions are languages which are meaningful to human but meaningless to computer. Take the example above, it had a field called number. The computer itself does not understand what “number” is (in this case the number printed on the player’s jersey). Therefore it still need human to process before the computer can provide the correct information for calling. There had been ongoing works (e.g. the Semantic Web) on building the Web that have content and services understandable to computer.

UDDI Basics

UDDI provides a place for web services to do the following functions:

- Let providers **publish** their businesses and web services provided to the registry
- Let client application **find** for some services or service providers
- Provide information on how a client can **bind** to the services found
There is two kind of UDDI: global and private. The global one is unique worldwide. Currently three companies – Microsoft, IBM and HP – are hosting the global UDDI. Businesses publish their service on any one of the operating nodes, the information would be propagated to all other nodes. These UDDI operators are free of charge and any businesses can publish their information onto it.

Companies or industries can also setup their own private UDDI. A private UDDI had some advantages over the global one. Examples are tighter control on the quality and accuracy of the services published and higher security on access control.

The data stored in the registry can be generally divided into three types:

- White pages – Name of business, contacts, descriptions, etc.
- Yellow pages – Information to classify the business like its nature and geographical location.
- Green pages – Technical information on the web services provided. It contains all the necessary information to find and bind to the services.

Also, the UDDI also allows registration of specific service type (called tModels in UDDI). Both service providers and consumers can refer to a specific type to implement and use the service.

Before a service can be published on a UDDI, it must first register a business entity. After a business record exists, it can publish their services to it. The UDDI specification strongly recommends all providers use WSDL to describe their services.
UDDI also have a set of API to publish and find business and services programmatically using SOAP. Therefore, the UDDI can be considered as a web services. This makes the whole set of protocols to be platform independent. Following is an example of request/response pair to a UDDI on searching for business with name started with “Stephon”.

Request
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <find_business xmlns="urn:uddi-org:api" generic="1.0" maxRows="100">
      <findQualifiers />
      <name>Stephon</name>
    </find_business>
  </Body>
</Envelope>

Response
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <businessList generic="1.0" xmlns="urn:uddi-org:api" operator="www.ibm.com/services/uddi" truncated="false">
      <businessInfos>
        <businessInfo businessKey="D1304F50-189B-11D6-A0DC-000C0E00ACDD">
          <name>Stephon FYP</name>
          <description xml:lang="en">All of the services here are fictitious</description>
          <businessService businessKey="D1304F50-189B-11D6-A0DC-000C0E00ACDD">
            <name>Hotel-USANYK-Hotel Marion</name>
          </businessService>
        </businessInfo>
      </businessInfos>
    </businessList>
  </Body>
</Envelope>
As seen above, all elements registered on a UDDI will be assigned a unique identification key. This is useful when searching for a specific set of records.
Design and Implementation

Methodology

The design of the web services implementation involves the core protocols, namely SOAP, WSDL and UDDI. The system is divided into two parts: service providers and service consumer.

For the providers’ part, four services are developed. They are hotel room and flight tickets reservation (fictitious), currency exchange rate and local temperature enquiry (dynamically fetch data from Internet). These services would allow application to call them using SOAP on HTTP. The hotel web services had also been published to an UDDI registry for dynamic discovery and binding using WSDL.

The consumer site is a web site called Travel Services, or TS for short. It integrates the above services into a user-friendly interface for end-user customer. This demonstrates a main characteristic of web services: they are modular applications that can be re-used and bind together to meet different needs.

Environment Settings and Tools Used

Two of the most popular platform to develop web services are the Microsoft .NET and Java. In this project, Java was chosen for the implementation language because of the following reasons:

- There exists more developer toolkits and library that are freely available for the Java language.
Most of the software is open source. This usually means faster development and can have a more in-depth understanding of the technology.

All of the applications are deployed on the same machine running Microsoft Windows 2000. Although it may look strange to perform distributed computing on the same machine, it is common for developing condition. It gives greater control over the system while without loss of generality.

The following libraries/tools are used in the implementation:

**Apache SOAP 2.2 for Java**
It is a library to manipulate SOAP message. It includes a web interface (as Java Servlet) for deployment into a web server to receive SOAP request.

**Apache Xerces Parser 1.44 and JAXP 1.0 (Java for XML Processing)**
They are used to parse and process XML documents. The Apache SOAP relies on them to process SOAP messages.

**Apache Tomcat 4.0**
It is used as the web server and Servlet container for the service consumer web site (client side).

**Sun J2EE SDK Server 1.3**
It is used to host and run service providers’ application. The applications are written with J2EE technology and exposes as web services.

**IBM Web Services Toolkit (WSTK) 2.4**
This toolkit contains a lot of useful tools and library to facilitate the development of web services. This project used its WSDL tools to generate the skeleton of the WSDL files and its UDDI library for Java to perform UDDI directory search. It also generates the client proxy class to access the web services.

**IBM Test UDDI Registry**
This registry is hosted by IBM on the Internet. It is specially used for developer to test their web services. The hotel reservation services had been published on this registry.
MySQL Relational Database 3.23 & its JDBC Driver
It is used to store information like booking information and user details.

System Structure

Here we define again clearly the three roles in this system:

- **Service provider**: They are applications that provide service or information, and exposed as web services.

- **Service consumer**: They are applications who use the content provided by web services. In this case it is the TS Web Site.

- **End-user customer**: They are the people who access the TS Web Site. They are said to be indirectly using the web services via the site.

Following are the detail of the structure for the service provider and consumer.

**Service Providers**
All of the services applications are implemented as Enterprise JavaBeans (EJB) on the J2EE server. Following is the structures for the different services.
Currency Exchange Rate and Temperature Enquiry
When a request is received, they first dynamically get data from the Internet. After some manipulation and filtering on the HTML, it will encode the information into the SOAP messages and sent back to the client.

These two services will provide WSDL files to define their interface. Clients who want to use them can refer to the WSDL files and create the client applications to access them. This step is only needed for the initial use. Also, they assume that the client would know their access point to use, so when calling them, the client need not to lookup to the UDDI.

Following is the summary of the methods in the two services:

- **Double GetExchangeRate(String from, String to)**
- **int getTemperature(String location)**

Hotel Room Reservation
For this service, it assumed a scenario that the hotel industry will first draft an interface that includes all of the necessary features and calling methods. After finalizing the interface, it will be publicly available as registered at the UDDI as a Service Type. Any hotel who wants to expose their online reservation as web services, they will need to comply with the interface file. After they have deployed their services that matched the interface definition, they will publish their services on the UDDI registry. They will also state that their service is implementing the Service Type above.

When a people want to reserve a hotel, it will specify a geographical location. Therefore, the UDDI registry must have some means to quickly filter out hotels outside that region so that the client would not waste time on contacting them. The UDDI specification allows services to set a service locator to indicate their geographical location by using Geographic Classification System (GCS) standard. However, this standard only specify up to the State level (e.g. Illinois, USA). This obvious is not very useful in this case. Therefore, this system uses its own method to divide into city level. It used six characters as the prefix for the service name to achieve that (e.g. USANYK = New York City, NY, USA). Note that this is only to show
a way to do the job. How the code can actually identify all cities is not the concern.

In the implementation, two fictitious hotels (both located in New York City) web services are setup and published to the UDDI. The two services are the same, except of the name of hotel and the service access point. The clone is to show how hotels can be dynamically found in runtime.

The function provided in this service includes room enquiry, room reservation and check previous transactions.

Following is the summary of the methods of this service:

- `int GetRating()`
- `String GetName()`
- `String GetAddress()`
- `int GetCheckInTime()`
- `int GetCheckOutTime()`
- `RoomDetail GetRoomDetail(Date checkin, Date checkout, String roomType, int minimumRent, int maximumRent, int quantity)`
- `BookDetail BookRoom(Date checkin, Date checkout, int quantity, String type, String cardNo, String passportNo, String customerName, int totalRent)`
- `BookDetail CheckBooking(String transactoinNo, String cardNo)`

The `RoomDetail` and `BookDetail` are self-defined data type structures. It contains information regarding room information and status and booking records respectively.

**Flight Tickets Reservation**
This service is only used to show how an application can use different services on the same application. Therefore, its structure is relatively simple and would not involve UDDI.

Following is the summary of the methods of this service:
- `String GetName()`

- `String GetPhone()`

- `FlightInfo FlightQuery(String departCountry, String departCity, String arrivalCountry, arrivalCity, Date depart, Date return, int quantity)`

- `TicketDetail BookTicket(Date depart, Date return, int quantity, String departCountry, String departCity, String arrivalCountry, arrivalCity, String cardNo, String passportNo, String name, int totalFare)`

Same as the hotel reservation, the two return type is self-defined that contain information of the flight.

**Service Consumer**

The TS site consists of JSPs (Java Server Pages), JavaBeans and other Java classes. JSP, like ASP and PHP pages, are used to create dynamic web pages. When a request is being made by a customer via JSPs, it will package the request into a SOAP messages and sent it to the service provider. The encoding of the SOAP messages is done by a web services proxy class, so that web designers are encapsulated from the web services technology. They can call the services like local procedures.

As mentioned above, the hotel reservation would involve lookup to UDDI at real time. The following paragraph describes how it works:

The site knows beforehand the t-model key (a unique identifier number) of the hotel reservation service type on the UDDI registry. It will first perform a search on all businesses that has provided services implemented that type. After that it will search for the service provided by the matched businesses that implement the type. Also, it will specify that only services names with prefix matching the selected region (see “Hotel Room Reservation” above) will be returned. After that, it will get all of the access points and relevant information of services that matched the criteria. Finally the site will contact each matching hotels and call their services.
Development

1. Setup servers and required libraries
First of all Java 2 Standard Edition (J2SE) and Java 2 Enterprise Edition (J2EE) were installed. Then all other required application and libraries were installed. After installing all packages, the web servers’ classpath need to be set to point to the installation paths of the libraries. Apache SOAP is installed into the web server as a component in order to accept SOAP requests.

2. Develop the first web services – temperature enquiry
As mentioned above, web services are independent of implementation language. It can be viewed as a wrapper to the application and exposes it to the public in the form web services. Therefore, the first thing to do is to write a Java program capable of performing the temperature enquiry. The program first creates a connection to the Hong Kong Observatory web site and retrieves the HTML file containing the required information. Then the program will extract the data from it and return the requested temperature.

After testing the program, it needs to be exposed as web services. This is done by deploying the service in the Apache SOAP. The Apache SOAP has a web interface for deploying web services.

To deploy this service, an ID that can uniquely identify was provided. Also the method names and the class name that want to expose as web services were entered. In this case there was only method called “getTemperature” and “com.weather.HKWeather” (class name for the program) respectively. The input and output parameters need not to be specified here. Since this program did not involve complex data types and structures, the other inputs were left empty.

After pressing the Deploy button in the bottom of the page, the services was deployed. It is now ready to be consumed.
Using the web interface can be time consuming, especially when it need to be redeploy often during development stage. The tool also provides command line programs to administrate it. All of the deployment settings can be saved to an XML file called deployment descriptor, so that there is no need to enter the information again and again.

As mentioned above, the calling details were not provided when deploying the web services. This is because services details and description are provided in a WSDL file. To generate the WSDL file, the WSDL generation tool provided in the IBM WSTK was used. The tool can read in a Java program and automatically generate WSDL files.

First, the class name and relevant binding information was provided to the tool. In the next screen, we can select the method we want to expose as web services. The WSDL is then generated.
The WSDL file can then be used on the client side proxy class generation since it describes all the details for calling the services. A command line tool takes the WSDL as its input parameter, and return necessary files to call the web services.

3. Convert the program to J2EE EJB architecture
J2EE platform provides more added functionality for enterprise like better XML support, easier modular application reuse and deployment and automatic concurrency control. Therefore, the program is converted to J2EE EJB application. Since the Apache SOAP does not yet provide suitable class to access EJB applications, a class for that purpose was written.

By packaging the applications into an Enterprise Application Archive (EAR), it can be easily transfer and deployed on any J2EE servers.

4. Create other web services
* Some source code for the program can be found in Appendix

The other services are developed similarly as above. Sample data are initialized for the database. Since the IBM WSTK tools does not support complex data types very well, the automatically generated classes need to be examined and refined before use.

Messages Format
In the implementation, all SOAP messages are communicate in client-server model. It did not involve any SOAP intermediaries (see SOAP
Basics of Chapter 2). Therefore, SOAP headers were not presented. All of the parameters are contained in the SOAP Body. Following is an example of a request-response pair of hotel room services enquiry:

**Request**

```
POST /soap/servlet/rpcrouter HTTP/1.0
Host: 127.0.0.1
Content-Type: text/xml; charset=utf-8
Content-Length: 977
SOAPAction: "urn:hotels"
Cookie: JSESSIONID=C2E4131533F7B505B1B13010630015E2

<?xml version='1.0' encoding='UTF-8'?><SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:GetRoomDetail xmlns:ns1="urn:hotels" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <checkin xmlns:ns2="http://proxy.hotels.com" xsi:type="ns2: Date">
        <year xsi:type="xsd:int">2002</year>
        <month xsi:type="xsd:int">4</month>
        <day xsi:type="xsd:int">23</day>
      </checkin>
      <checkout xmlns:ns3="http://proxy.hotels.com" xsi:type="ns3: Date">
        <year xsi:type="xsd:int">2002</year>
        <month xsi:type="xsd:int">4</month>
        <day xsi:type="xsd:int">25</day>
      </checkout>
      <room_type xsi:type="xsd:string">SG</room_type>
      <min_rent xsi:type="xsd:int">0</min_rent>
      <max_rent xsi:type="xsd:int">300</max_rent>
      <quantity xsi:type="xsd:int">1</quantity>
    </ns1:GetRoomDetail>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

As seen above, the SOAP message is transmitted over HTTP POST. All of the parameters are self-explanatory. The response message is listed below:

**Response**

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 1082
Date: Fri, 19 Apr 2002 08:30:14 GMT
Server: J2EE SDK/1.3 (HTTP/1.1 Connector)

<?xml version='1.0' encoding='UTF-8'?><SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
```

---

**chapter 3 : design and implementation**

28
Screenshots

This section will show a sample run on the TS site.

When the user clicked on the “Hotels” link on the left, it asks user to input certain criteria. The user can at the same time reserve round-trip tickets at the same time. After finishing, click the “Search Now” button to continue.
The system would then display the result of the enquiry. This page consumed a total of three web services. They are:

1. Enquiry of hotel room availability
2. Enquiry of flight tickets availability
3. Currency exchange rate (US Dollars to HK Dollars in this case)

After the user confirms their bookings and login, the transaction is done by calling the respectively methods of the web services and the result would be display on screen.
The screenshots below show the temperature enquiry and currency rate.
Analysis

Performance
From the customer point of view, the service consumer part of the system (TS site) provides a one-stop shop for them. As this involves calling different remote systems, the waiting time may vary due to factors like network traffic and system availability.

Development Efficiency
The above performance advantages can well be achieved by other means like integration of technology between participating partners and using proprietary communication technologies. Therefore, the largest gain is the development efficiency between the service providers and consumers. They can develop their applications faster and easier by using a standard framework that is acceptable worldwide. Also application can plug in different services as their building components easily.

In the implementation, several tools were used to automate steps like WSDL and client proxy generation in the development process. This would be useful and can reduce the development time greatly. However, we should have a good understanding of what is doing behind the scene. This can help us to optimize the generated codes and debugging.

Interoperability
Although the whole implementation is running under Java environment, it can safe to say that it do have a high level of interoperability. This is because all of the communications are done by using standard protocols like XML, SOAP and WSDL. These protocols are defined by standard bodies...
that are implemented in different platforms. Since all developer needs to follow the structure of them, different platform should not have a big problem in calling others.

Actually, a simple program had been written in Java to call a simple web service provided on the Internet, which is implemented by Microsoft .NET platform. The return result is the same as using the client offered by the service provider.

Since the web services technology is still under development, there are unavoidably some kind of incompatibility exists between systems. However, those issues should be minor and should not affect most of the users.

**Difficulties**

- Originally, a private UDDI was planned to setup for the hotel web services. However, the lite version of the UDDI Registry bundled with the IBM WSTK was unable to run successfully after trying different environment settings. Therefore, the IBM Test UDDI Registry was used instead. It provides all the functionality from the UDDI specification and a convenient web interface. Like the global UDDI, it is free of charge and is globally accessible.

- The Apache SOAP did not have a suitable class for communicating with the EJB deployed in the J2EE SDK Server. To cope with the problem, a Java class was written by referring to the source codes and documents provided with the Apache SOAP toolkit.

- Since web services is still under development, many libraries and toolkits used in the project had new versions comes out quickly. Some of them may become obsolete in a few months. Since this project aims is to study the overall framework that is well-defined, the implementation system environment was stick to the original settings.
Improvements

Security
In this system, the security issue was not concerned. In order for real-life deployment, security would be an important issue. Measures like transporting over HTTPS, applying web services security extensions and digital certification are some of the possible way to improve the system security.

Message Passing
Since the system involves only two hotel services providers, the waiting time is acceptable for real-time enquiry. However in real-life situation, the servers are far away and the numbers of services to contact are far more than two. In that sense, it may not be feasible to do RPC calls to all of them and then display the results in the same run. Therefore, the system can be implemented as messaging passing and it will in turn inform the customer (for example thru email) when all of the results are ready. The customers can then go back and continue make appropriate choice.
Conclusion

After finishing this project, I have gained in-depth knowledge on Web Services and how it works as a whole. Since the core protocols of Web Services, namely SOAP, WSDL and UDDI, are based on XML and are open standard, it open a new way of doing distributed computing over the Internet. Businesses can share their information and carry out transactions more efficiently.

Since Web Services are also self-describing, applications can have a high degree of automation. Many functions can be performed without human intervention. Combining with the dynamic discovery by UDDI, applications can locate services in real-time.

In the implementation of the system, I have got practical experience of setting up servers and deploying Web Services on the Java platform.


Selected Source Codes

Currency Exchange Rate Services

Server Application (EJB)
* Remote and Remote Home Interface of EJB are intentionally omitted for clarity.

//currency.java
package com.currency;

import java.rmi.*;
import javax.ejb.*;
import java.io.*;
import java.net.*;

public class Currency implements SessionBean {
    private SessionContext sessionContext = null;

    public Currency() {};
    public void ejbCreate() {};
    public void ejbRemove() throws EJBException, RemoteException {};
    public void ejbActivate() throws EJBException, RemoteException {};
    public void ejbPassivate() throws EJBException, RemoteException {};
    public void setSessionContext(SessionContext context) throws EJBException, RemoteException {
        this.sessionContext = context;
    }

    // If debug mode is set to true, it will return a dummy value
    // without accessing the Internet to get real time information.
    public double GetExchangeRate(String from, String to, boolean debug) {
        Socket s;
        BufferedReader in;
        PrintWriter out;
        String line, query, s_result;
        double result = 0.0;
        boolean found = false;
        query = "convert/classic?user=bball&value=100&exch=\"from\"+\"expr\"+\"to;"
        if (!debug) {
            try {
                s = new Socket("www.oanda.com",80);
                in = new BufferedReader(new InputStreamReader(s.getInputStream()));
                out = new PrintWriter(s.getOutputStream(),true);
                out.println("GET "+query+" HTTP/1.1");
                out.println("Host: www.oanda.com");
                out.println("\"");
                while (!found) {
                    line = in.readLine();
                    if (line.indexOf("Median price")!=-1) {
                        s_result = line.substring(line.indexOf("\")+1,line.indexOf("(bid/ask)"));
                        result = Double.parseDouble(s_result);
                        found = true;
                    } else if (line.indexOf("Error")!=-1) {  
                        System.out.println("Error: "+line);
                    }  
                }
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                try {
                    in.close();
                    out.close();
                    s.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        } else {
            s_result = "dummy value";
        }
        return result;
    }

    // Example usage of GetExchangeRate
    public void main(String[] args) {
        double rate = GetExchangeRate("USD", "EUR", true);
        System.out.println("The current rate is "+rate);
    }
}
Deployment Descriptor

```xml
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
    id="urn:currency" checkMustUnderstands="false">
    <isd:provider type="java" scope="Application" methods="GetExchangeRate">
        <isd:java class="com.currency.Currency" static="false"/>
    </isd:provider>
</isd:service>
```

WSDL

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Currency"
    targetNamespace="http://www.currencyservice.com-interface"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
    xmlns:tns="http://www.currencyservice.com-interface"
    xmlns:types="http://www.currencyservice.com-interface/types/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <message name="GetExchangeRateRequest">
        <part name="from" type="xsd:string"/>
        <part name="to" type="xsd:string"/>
        <part name="debug" type="xsd:boolean"/>
    </message>

    <message name="GetExchangeRateResponse">
        <part name="rate" type="xsd:double"/>
    </message>

    <portType name="Currency">
        <operation name="GetExchangeRate">
            <input message="tns:GetExchangeRateRequest"/>
            <output message="tns:GetExchangeRateResponse"/>
        </operation>
    </portType>

    <binding name="CurrencyBinding" type="tns:Currency">
        <soap:binding style="rpc"
            transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetExchangeRate">
            <soap:operation
                soapAction="urn:currency"/>
            <input>
```
<soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  namespace="urn:currency" use="encoded"/>
</input>
<output>
<soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  namespace="urn:currency" use="encoded"/>
</output>
</operation>
</binding>

<service name="Currency">
  <port binding="CurrencyBinding" name="CurrencyPort">
    <soap:address location="http://localhost:8000/soap/servlet/rpcrouter"/>
  </port>
</service>
</definitions>

Client Proxy Class

//Binding.java
package com.currency.proxy;

public class Binding {

  public static java.net.URL[] KnownServiceLocations = null;
  private org.apache.soap.rpc.Call call = new org.apache.soap.rpc.Call();
  private java.net.URL url = null;
  private java.lang.String SOAPActionURI = ""
  private org.apache.soap.encoding.SOAPMappingRegistry smr =
  call.getSOAPMappingRegistry();
  static
  {
    try{
      KnownServiceLocations = new java.net.URL[]{
        new java.net.URL("http://localhost:8000/soap/servlet/rpcrouter")
      };
    }catch(java.net.MalformedURLException e ){
      KnownServiceLocations = new java.net.URL[0];
    }
  }

  public Binding()
  {
    call.setTargetObjectURI("urn:currency");
    call.setEncodingStyleURI("http://schemas.xmlsoap.org/soap/encoding/");
    this.url = KnownServiceLocations[0];
    this.SOAPActionURI = "urn:currency";
  }
  
  public Binding(java.net.URL endPointURL)
  {
    call.setTargetObjectURI("urn:currency");
    call.setEncodingStyleURI("http://schemas.xmlsoap.org/soap/encoding/");
    this.url = endPointURL;
    this.SOAPActionURI = "urn:currency";
  }
}
public synchronized double GetExchangeRate (java.lang.String from, java.lang.String to, java.lang.Boolean debug) throws org.apache.soap.SOAPException
{
    if(url == null) throw new org.apache.soap.SOAPException(org.apache.soap.Constants.FAULT_CODE_CLIENT, "A URL must be specified \"CurrencyBinding\".");
    java.util.Vector parms = new java.util.Vector(2);
    parms.addElement( new org.apache.soap.rpc.Parameter("from", java.lang.String.class, from, null));
    parms.addElement( new org.apache.soap.rpc.Parameter("to", java.lang.String.class, to, null));
    this.call.setParams(parms);
    org.apache.soap.rpc.Response resp = this.call.invoke(url, SOAPActionURI);
    if(resp.generatedFault())
    {
        org.apache.soap.Fault fault = resp.getFault();
        throw new  org.apache.soap.SOAPException(fault.getFaultCode(), fault.getFaultString());
    }
    return ((java.lang.Double)resp.getReturnValue().getValue()).doubleValue();
}

Hotel Reservation Services

Server Application (EJB)
* Remote and Remote Home Interface of EJB are intentionally omitted for clarity.

//Hotel.java
package com.MHB;
import java.rmi.*;
import javax.ejb.*;
import java.sql.*;
import java.util.*;
public class Hotel implements SessionBean {

    private SessionContext sessionContext = null;
    private static final String name = "Marion Hotel";
    private static final String address = "17 Kennedy Road, Richmond Hill, New York, NY, USA";
    private static final int rating = 4;
    private static final int checkInTime = 1400;
    private static final int checkOutTime = 1130;
    private static final String databaseURL = "jdbc:mysql://127.0.0.1:3306/fyp_hotel01";

    public Hotel() {}
    public void ejbCreate() {}
    public void ejbRemove() throws EJBException, RemoteException {}
    public void ejbActivate() throws EJBException, RemoteException {}
    public void ejbPassivate() throws EJBException, RemoteException {}
}
public void setSessionContext(SessionContext context) throws EJBException, RemoteException {
    this.sessionContext = context;
}

public int GetRating() {
    return rating;
}

public String GetName() {
    return name;
}

public String GetAddress() {
    return address;
}

public int GetCheckInTime() {
    return checkInTime;
}

public int GetCheckOutTime() {
    return checkOutTime;
}

public com.hotels.RoomDetail[] GetRoomDetail(com.hotels.TheDate from, com.hotels.TheDate to, String type, int lowPrice, int highPrice, int quantity) {
    com.hotels.RoomDetail rd[] = null;
    Vector result = new Vector();
    java.sql.ResultSet rs = null;
    String sql;
    int normalPrice, holidayPrice, totalPrice, dateDiff, holidayCount;

    sql = "SELECT MIN(available), normal_price, holiday_price ";
    sql += "FROM room_status s, room_price p ";
    sql += "WHERE s.type=" + type + " AND date>='" + from.GetYear() + "-" + from.GetMonth() + "-" + from.GetDay() + "' ";
    sql += "AND date<'" + to.GetYear() + "-" + to.GetMonth() + "-" + to.GetDay() + "' ";
    sql += "AND normal_price>=" + lowPrice + " AND normal_price<=" + highPrice + " AND s.type=p.type ";
    sql += "GROUP BY s.type";
    dateDiff = DateDiff(from, to);
    holidayCount = HolidayCount(from, to);
    rs = DBAccess(sql);
    try {
        while (rs.next()) {
            if (rs.getInt(1) >= quantity) {
                normalPrice = rs.getInt(2);
                holidayPrice = rs.getInt(3);
                totalPrice = quantity * (normalPrice*(dateDiff-holidayCount) + holidayPrice*holidayCount);
                result.addElement(new com.hotels.RoomDetail(from, to, totalPrice, type, "No description available."));
            }
        }
    } catch (Exception ex) { System.err.println(ex); }
    rd = new com.hotels.RoomDetail[result.size()];
    for (int i=0; i<result.size(); i++)
        rd[i] = (com.hotels.RoomDetail)result.elementAt(i);
public com.hotels.BookDetail BookRoom(com.hotels.TheDate from, com.hotels.TheDate to, int quantity, String type, String cardNo, String nationality, String passportNo, String customerName, int _totalPrice) {

    String sql, tranNo;
    int normalPrice, holidayPrice, totalPrice, dateDiff, holidayCount;
    boolean available, success;
    com.hotels.BookDetail bd;
    java.sql.ResultSet rs = null;

    totalPrice = 0;
    available = success = false;
    sql = "SELECT MIN(available), normal_price, holiday_price ";
    sql += "FROM room_status s, room_price p ";
    sql += "WHERE s.type='" + type + "' AND date>='" + from.GetYear() + "-" + from.GetMonth() + "-" + from.GetDay() + "' ";
    sql += "AND date<='" + to.GetYear() + "-" + to.GetMonth() + "-" + to.GetDay() + "' ";
    sql += "AND s.type=p.type ";
    sql += "GROUP BY s.type";
    dateDiff = DateDiff(from, to);
    holidayCount = HolidayCount(from, to);
    rs = DBAccess(sql);

    // Check for availability and price
    try {
        while (rs.next() && !available)
            if (rs.getInt(1) >= quantity) {
                normalPrice = rs.getInt(2);
                holidayPrice = rs.getInt(3);
                totalPrice = quantity * (normalPrice * (dateDiff - holidayCount) + holidayPrice * holidayCount);
                if (totalPrice == _totalPrice)
                    available = true;
            }
    } catch (Exception ex) { System.err.println(ex); }

    // Reserve the requested rooms
    if (available) {
        sql = "UPDATE room_status SET available=available-" + quantity;
        sql += " FROM room_status s, room_price p ";
        sql += "WHERE type='" + type + "' AND date>='" + from.GetYear() + "-" + from.GetMonth() + "-" + from.GetDay() + "' ";
        sql += "AND date<='" + to.GetYear() + "-" + to.GetMonth() + "-" + to.GetDay() + "' ";
        sql += "AND s.type=p.type ";
        sql += "GROUP BY s.type";
        DBAccess(sql);
        tranNo = GenerateTransactionNo();
        sql = "INSERT INTO book_record VALUES('" + tranNo + "'," + customerName + "," + passportNo + "," + cardNo + "," + from.GetYear() + "-" + from.GetMonth() + "-" + from.GetDay() + "," + type + "," + quantity + "," + totalPrice + ")";
        DBAccess(sql);
        bd = new com.hotels.BookDetail(from, to, totalPrice, quantity, cardNo, type, "No desc.", customerName, passportNo, tranNo);
    }
}

return rd;
}
else
    bd = new com.hotels.BookDetail(false);
    return bd;
}

public com.hotels.BookDetail CheckBooking(String tranNo, String cardNo) {
    String sql;
    java.sql.ResultSet rs = null;
    com.hotels.BookDetail bd = new com.hotels.BookDetail(false);
    sql = "SELECT ref_no,name,passport_no,card_no,checkin,checkout,type,quantity,rent FROM book_record ";
    sql += "WHERE ref_no='" + tranNo + "' AND card_no='" + cardNo + "'";
    rs = DBAccess(sql);
    try {
        if (rs.next()) {
            bd = new com.hotels.BookDetail(Str2Date(rs.getString(5)),Str2Date(rs.getString(6)),rs.getInt(9),rs.getInt(8),rs.getString(4),rs.getString(7),"No desc.",rs.getString(2),rs.getString(3),rs.getString(1));
        }
    } catch (Exception ex) { System.err.println(ex); } 
    return bd;
}

private java.sql.ResultSet DBAccess(String sql){
    Connection con = null;
    java.sql.ResultSet rs = null;
    try {
        Class.forName("org.gjt.mm.mysql.Driver");
    } catch(java.lang.ClassNotFoundException e) { 
        System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());
    }
    try {
        con = DriverManager.getConnection(databaseURL, "fyp", "fyp");
    } catch(SQLException ex) { 
        System.err.println("init - SQLException: "+ex.getMessage());
    }
    try {
        Statement stmt = con.createStatement();
        rs = stmt.executeQuery(sql);
    } catch(SQLException ex) { 
        System.err.println("SQLException: " + ex.getMessage());
    }
    return rs;
}

private String GenerateTranscationNo() {
    String no;
    Calendar c1 = Calendar.getInstance();
    no = "" + c1.get(c1.SECOND) + c1.get(c1.YEAR) + c1.get(c1.HOUR) + c1.get(c1.MONTH) + c1.get(c1.MINUTE) + c1.get(c1.DATE);
    no += (int)(Math.random()*99+1);
    return no;
}
Deployment Descriptor

```xml
<?xml version="1.0"?>
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
  id="urn:hotels" checkMustUnderstands="false">
  <isd:provider type="MyStatelessEJBProvider"
    scope="Application"
    methods="GetRating GetName GetAddress GetCheckOutTime GetCheckInTime GetRoomDetail BookRoom CheckBooking"
    <isd:java class="com.MHB.Hotel"/>
    <isd:option key="FullHomeInterfaceName" value="com.MHB.HotelRemoteHome"/>
    <isd:option key="JNDIName" value="HotelMarion"/>
  </isd:provider>
  <isd:mappings>
    <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:x="http://proxy.hotels.com" qname="x:TheDate"
      javaType="com.hotels.TheDate"
      xml2JavaClassName="com.hotels.proxy.TheDateSerializer"
      java2XMLClassName="com.hotels.proxy.TheDateSerializer"/>
    <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:x="http://proxy.hotels.com" qname="x:RoomDetail"
      javaType="com.hotels.RoomDetail"
      xml2JavaClassName="com.hotels.proxy.RoomDetailSerializer"
      java2XMLClassName="com.hotels.proxy.RoomDetailSerializer"/>
    <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:x="http://proxy.hotels.com" qname="x:BookDetail"
      javaType="com.hotels.BookDetail"
      xml2JavaClassName="com.hotels.proxy.BookDetailSerializer"
      java2XMLClassName="com.hotels.proxy.BookDetailSerializer"/>
  </isd:mappings>
  <isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
</isd:service>

WSDL
(Interface WSDL, not wholly listed)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="hotels" targetNamespace="http://www.hotels.com"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.hotels.com"
  xmlns:types="http://www.hotels.com-interface/types/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <types>
    <xsd:schema targetNamespace="http://www.hotels.com/types/"
      xmlns="http://www.w3.org/2001/XMLSchema/">
      <xsd:complexType name="RoomDetail">
        <xsd:sequence>
          <xsd:element name="totalPrice" type="xsd:int"/>
          <xsd:element name="descr" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="ArrayOfRoomDetail">
        <xsd:complexContent>
          <xsd:restriction base="SOAP-ENC:Array">
          </xsd:restriction>
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:schema>
  </types>
</definitions>

appendix : selected source codes 44
Client-side JSPs

Temperature Enquiry
Text in grey refers to codes calling the web services (using the client proxy class listed above)

<html>
<jsp:useBean id="country" scope="page" class="com.country.Country" />
<%@ page session="true" %>
<% String from, to;
double rate = 0, amount=0, result = 0;
boolean success = false;
com.currency.proxy.Binding binding;
binding = new com.currency.proxy.Binding();
from = request.getParameter("from");
to = request.getParameter("to");
try {
    amount = Double.parseDouble(request.getParameter("amount"));
    rate = binding.GetExchangeRate(from, to, false);
} catch (Exception ex) {} 
if (rate!=0)
    success=true;
result = ((int)(amount*rate*1000))/1000.0;
%
<head>
<title>Travel Service - Currency Converter</title>
...
</html>

**UDDI Enquiry code**

This class is used to search the UDDI for the hotel reservation services

```
package com.UDDI;

import java.net.*;
import java.util.*;
import java.io.*;
import javax.xml.parsers.*;
import com.ibm.uddi.*;
import com.ibm.uddi.client.*;
import com.ibm.uddi.response.*;
import com.ibm.uddi.datatype.*;
import com.ibm.uddi.datatype.tmodel.*;
import com.ibm.uddi.datatype.business.*;
import com.ibm.uddi.datatype.service.*;
import com.ibm.uddi.datatype.binding.*;
import com.ibm.uddi.util.*;
import org.apache.soap.*;
import org.apache.soap.transport.http.SOAPHTTPConnection;
import org.apache.xml.serialize.*;
import org.xml.sax.InputSource;

public class SearchUDDI {
    private UDDIProxy uddiproxy = new UDDIProxy();
    private BusinessList bl = null;
    private Vector blv = new Vector();
    private ServiceList sl = null;
    private Vector slv = new Vector();
    private ServiceDetail sd = null;
    private Vector sdv = new Vector();
    private TModelList tml = null;
    private TModelBag tmb = null;

    /* This function takes the UDDI Inquiry URL and a vector containing the service type tModelKey that want to be searched. A search will first performed to search for businesses implementing the service type (find_business). After that, it search for the service provided by the matched business that implement the
```
service type (find_service). Finally, it drill down to get the access point and its relevant info of each service implementation (get_serviceDetail). */

    public HotelUDDIResult HotelFindServiceByTModelKey(String inqURL, Vector tmv, int debugMode) {
        int i, j;

        HotelUDDIResult infov = new HotelUDDIResult();
        if (debugMode==0) {
            try {
                tmb = new TModelBag(tmv);
                uddiproxy.setInquiryURL(inqURL);
                bl = uddiproxy.find_business(tmb,null,0);
                blv = bl.getBusinessInfos().getBusinessInfoVector();
                for (i=0; i<blv.size(); i++) {
                    sl  = uddiproxy.find_service(((BusinessInfo)blv.elementAt(i)).getBusinessKey(),tm b,null,0);
                    slv = sl.getServiceInfos().getServiceInfoVector();
                    for (j=0; j<slv.size(); j++) {
                        sd = uddiproxy.get_serviceDetail(((ServiceInfo)slv.elementAt(j)).getServiceKey() );
                        sdv.addAll (sd.getBusinessServiceVector());
                    }
                }
            }
            catch (Exception ex) { System.err.println(ex); }
            for (i=0; i<sdv.size(); i++) {
                infov.region.addElement(((BusinessService)sdv.elementAt(i)).getNameString().substring(6,12));
                infov.name.addElement(((Description)((BusinessService)sdv.elementAt(i)).get DescriptionVector().elementAt(0)).getText());
                infov.accessURL.addElement(((BindingTemplate )((BusinessService)sdv.elementAt(i)).getBindingTemplates().getBindingTemplateVector().elementAt(0)).getAccessPoint().getText());
            }
        } else if (debugMode==1) {
            ...
        }
        return infov;
    }