*Secure Web Development Teaching Modules*[1]

# Introduction to Web Services

## Contents



**Prerequisite**: Completion of SWEET Security Lab Series module "Introduction to Web Technologies".

# 1 Concepts

Since you have read, hopefully worked out, the *SWEET Security Lab Series* module "Introduction to Web Technologies", you should now have basic understanding of the four-tier web architecture, HTML form and the HTTP protocol for communications between web servers and web browsers. If you are not sure, please review that module before continue.

This module addresses the following fundamental challenges to today's computing industry: (a) how to let heterogeneous information systems communicate with each other, (b) how to let the consumers get computing services without the need of managing their own hardware/software systems, (c) how to

mechanically convert a net-blind legacy application into a net-aware application and deliver services across the Internet. The solution to all of these challenges depends on the technology called *web services*.

## 1.1 Challenges that Web Services Address

### *1.1.1 Integration of Heterogeneous Information Systems*

The business processes of a company are normally supported by many types of information systems, including transaction servers, inventory systems, accounting systems, and payroll systems. These systems may be from different companies and built with different technologies. For smooth and efficient operation of the company, these internal information systems need be integrated to work together.

For many companies, their operating and management units are distributed across multiple buildings, cities, or even across the world. For efficient global decision making and business coordination, the information systems of these distributed units must be integrated. Depending on the size, location, inception time, and function of these units, their information systems may differ in terms of operating platforms and technologies.

In today's business environment, very few companies can be really self-sufficient. Each company needs to consume products or services from other companies, and produce new products or services to serve its own clients. As a result, the Business-to-Business (B2B) computing model is very important, and the information systems of different companies need to be integrated to some extent so that they can conduct business transactions automatically without extensive human interventions. Information systems of different companies are usually based on different platforms and technologies.

The web service technology introduced in this module can solve this enterprise information system integration problem.

**Question 1:** What are the main advantages of hosting computing services on data centers and deliver them to web browsers through the Internet?

**Question 2:** What is the main difference of the web services from the computing services that we discussed in question 1?

### *1.1.2 Service-Oriented Architecture (SOA)*

*Service-oriented architecture (SOA)* is a style of software architecture for loose-coupling of software agents (programs or information systems of various granularities) in a distributed environment. A *service* could be as simple as a remote method invocation, or it could be a complete business transaction like buying a book from *Amazon* online. The services could also be classified into *horizontal* ones and *vertical* ones, the former providing basic common services to many other services' implementations, and the latter providing services in a particular application domain. The *service provider* and *service consumer* are two roles that could be played by different or the same software agent. The important services should have standardized *application programming interfaces (API)* so that a service consumer can choose from many service implementations conforming to the same standardized API.

The success of SOA depends on a few things. First, a standard interfacing technology must be adopted by all participating software agents, and service consumers should not need to install special tools to use

each special service implementation. Second, the service providers should have a standardized mechanism to publicize their services as to their availability, functions, APIs, cost, and quality assurance levels; and the service consumers could use the same mechanism to search for desired service implementations based on the industry category of the services in demand. A service consumer should be able to easily switch from one service implementation to another more cost-effective one. The standardization of the interfacing technology and business service API are critical to the success of SOA.

As an architectural style, SOA involves more than IT technologies. SOA could have impacts on business models or processes too. In this module we focus on the foundation problem of SOA: how to use web services to support communications between heterogeneous computing systems (software agents).

**Question 3:** Why *web service* API standardization is important to the success of the service-oriented architecture methodology?

**Question 4:** Why *service* API standardization is important to the success of the service-oriented architecture methodology?

### *1.1.3 Fast Transformation of Legacy Applications for E-Commerce*

Many enterprise applications take decades to improve and stabilize. They have been the cornerstones of the business operation. Most of these applications are designed for interacting with the company staff only, not with the end users whom they serve. We say such legacy applications are net-blind since they cannot communicate with the remote end users over the Internet.

With today's e-commerce and the ever-increasing communication models including social networking, the companies have the urge to provide similar services over the Internet to the national or global end users. In this environment the applications need to communicate directly with the end users and deliver many services with minimal staff involvement. It is just too expensive or slow to redesign and re-implement those mature applications, and even small changes to those applications could introduce subtle bugs. The solution is to use web service as a wrapper technology to transform the legacy applications and allow them communicate directly with the remote users without modifying the business logic core of the applications. This is one of the major strengths of web services.

**Question 5:** Why should you try to avoid rewriting programs for core business logics?

## 1.2 Fundamental Concepts of Web Services

Now we use a simple scenario to explain the fundamental ideas of web services. Assume computer A runs a Java object *Business* (business logic implementation) on the Linux, and the object supports a method with signature "String hello(String n)" (the invoker needs to send method "hello" a string, and the method then returns another string); and computer B runs a C++ object *Client* on the Windows; both computers are connected by some computer network including the Internet or a home LAN; now the object *Client* on computer B needs to invoke the method "hello" of object *Business* on computer A over the network. We need to address the following challenges:

1. How for computer A to accept method invocation requests over the network? The solution is to run a web server on computer A, and deploy a Java servlet or ASP .NET web page on the web server as the reception point web object for method invocations.

2. How for computer A to tell computer B the signature of method "hello" in a programming language neutral way (computer A may have no clue about the programming language in which method "hello" on computer B is implemented), and how for computer A to receive method invocation requests over the network? The solution is to define a simple XML dialect called *Web Service Description Language (WSDL)*, which is just expressive enough to specify the method signatures, like "String hello(String n)", and the URL for the reception point web object for method invocation (a Java servlet or ASP .NET web page on computer A's web server). A web service utility on computer A could read the source code of object *Business*, generate a WSDL file for describing method "hello", generate the reception point web object for method "hello", and deploy the reception point web object on the local web server. The WSDL file for method "hello" could be sent to computer B in any possible way including web downloading or email attachment. The following is the simplified WSDL file for method "hello".

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions>
  <message name="helloResponse">
    <part  name="helloReturn"  type="xsd:string" />
  </message>
  <message name="helloRequest">
    <part name="n" type="xsd:string" />
  </message>
  <portType name="HelloServer">
    <operation  name="hello"  parameterOrder="n">
      <input message="helloRequest"  name="helloRequest" />
      <output message="helloResponse"  name="helloResponse" />
    </operation>
  </portType>
  <service name="HelloServerService">
    <port binding="HelloServerSoapBinding" name="HelloServer">
      <address location="http://localhost:8080/axis/HelloServer.jws" />
    </port>
  </service>
</definitions>
```

3. How could computer B send method invocation arguments to computer A? The solution is to define a separate XML dialect *Simple Object Access Protocol (SOAP)* for specifying all information about a method invocation: name of the target method, and value for each of the parameters; or the return value of such a method invocation, in a programming language independent way (in XML). A simplified sample SOAP message may read like

```
<?xml version="1.0"?>
<Envelope>
  <Body xmlns:m="localhost:8080/axis/HelloServer.jws">
    <m:HelloServer>
       <m:helloRequest>John</m:helloRequest>
    </m:HelloServer>
  </Body>
</Envelope>
```

4. How could C++ object *Client* on computer B invoke the remote Java method "hello"? A web service utility, maybe called *wsdl2cpp*, on computer B would download the WSDL file for "hello", read it and generate source code for a new proxy object in a programming language specified by the developer to the utility. In our case a set of C++ source code would be generated to implement the proxy class. The proxy class supports a method "hello" with exactly the same

signature (method name and parameter number and data types) as the "hello" method on computer A but in C++ syntax. At run time the *Client* object would create a local proxy object and invoke its "hello" method. The proxy's method "hello" would convert the argument string into the neutral XML format, use this XML format argument and method name "hello" to generate a SOAP message, and initiate an HTTP POST request to the reception point web object on computer A (the URL is part of the WSDL file) with the SOAP message in the HTTP request's entity body. On computer A the reception point web object would pick up the HTTP request attachment, the SOAP message, unwrap it, convert argument value from XML into Java format, invoke the local Java *Business* object method "hello" with the converted argument from computer B, convert the return value into XML, wrap the converted return value in a new SOAP message, and return the SOAP message back to computer B as part of its HTTP response. Upon receiving this new SOAP message from computer A, the proxy object's method "hello" would unwrap the incoming SOAP message, convert the return value from XML format to C++ format, and return the resulting value as its own return value.

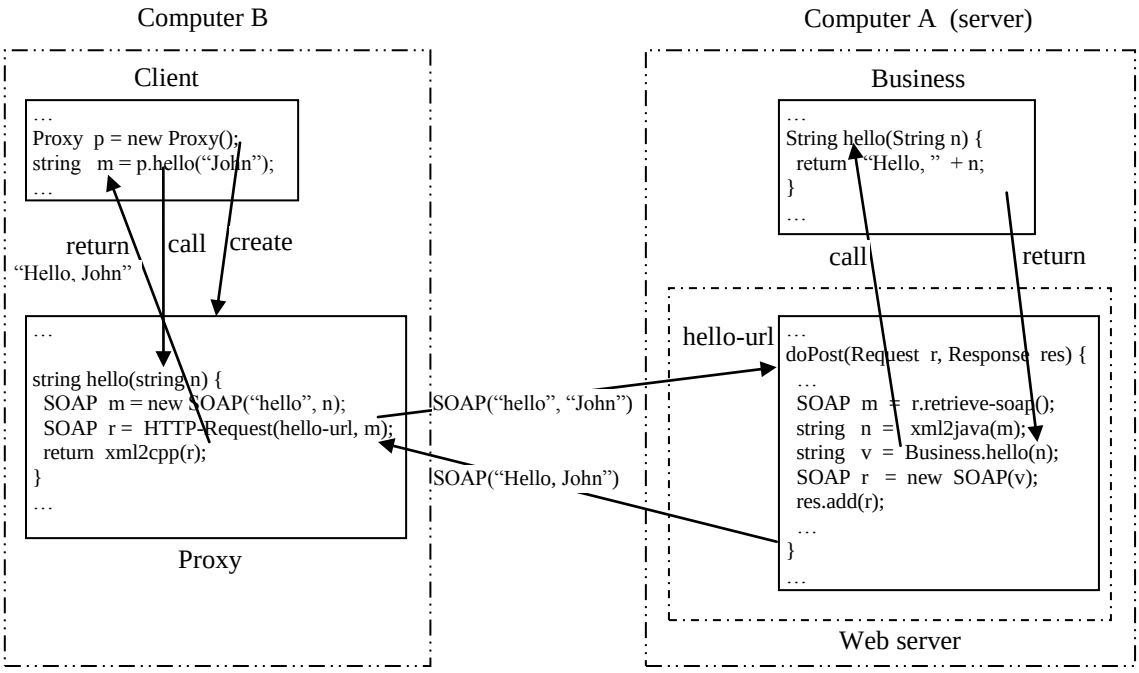



**Figure 1 Web service method invocation**

The above simple scenario explained the basic mechanism of web services. As with all remote procedure or method invocation technologies, web services is based on the popular *proxy design pattern* (http://sourcemaking.com/design_patterns/proxy).

One observation can be made that, since WSDL files and SOAP files are generated and consumed by tool utilities or machine-generated code, it is not mandatory for developers to understand completely the syntaxes of the WSDL and SOAP structures, even though some basic understanding of WSDL and SOAP is helpful. WSDL and SOAP are not intended for human beings to write or read. That is why even though they are important in the implementation of web services, they are not covered beyond logical descriptions in most textbooks on web services.

**Question 6:** What is the function of a WSDL file?

**Question 7:** What is the function of a SOAP message?

**Question 8:** What is the function of XML in web services?

**Question 9:** Do all servers running web services have a web server installed?

**Question 10:** What is the proxy design pattern?

## 1.3 Major Components of Web Services

Web services are based on the following major technologies:

- *Universal Description, Discovery and Integration (UDDI)*, a standard for web service registries to which web service providers can register and advertise their services, and potential clients can use the registries as yellow pages to search for potential service providers and download the WSDL service description files for the chosen services. UDDI allows programs to access the registries through their open APIs.

  At this point there are no commercial UDDI registries available. Microsoft, IBM and SAP used to run their public free UDDI registries to validate the technology, but they have shut down this service in 2006 after they declared that the UDDI validation was a success. The following web pages contain some public web services that you can explore and test.

  http://www.xmethods.com/ve2/index.po
  http://www.service-repository.com/
  http://seekda.com

- *Web Service Description Language (WSDL)*, a dialect of XML for specifying in a language-independent way the exposed web service methods' signatures, the data types that they depend on, the exceptions that the methods may throw at execution time, and the reception point on a web server for clients to send SOAP messages through HTTP POST/GET requests representing method invocations. Because WSDL files are generated and consumed by tools and machine-generated code, a web service developer does not need to write them or read them.

- *Simple Object Access Protocol (SOAP)*, a dialect of XML for specifying in language-independent way all information about a method invocation, including the name of the method, the arguments of the method invocation, return value, and return exceptions. Because SOAP messages are generated and consumed by tools and machine-generated code, a web service developer does not need to write them or read them.

- *Extensible Markup Language (XML)*, the base language for WSDL and SOAP. XML is not needed explicitly in the development of web services.

- *HyperText Transfer Protocol (HTTP)*, a common carrier for sending SOAP messages between web service client systems and service implementations. We need to have a basic understanding of how a web server processes the HTTP POST/GET requests.

## 1.4 Web Service's Role in System Integration

If you are sure that your application and its consuming applications, local or remote, are all based on the Windows platform, then Microsoft .NET remoting (http://msdn.microsoft.com/en-us/library/ms973857.aspx), part of Microsoft COM+, is the best technology for system integration because it is customized for the Windows platform. On the other hand, if you are sure that your application and its consuming applications, local or remote, are all based on the Java platform, then Java Remote Method Invocation over IIOP, RMI-IIOP (http://en.wikipedia.org/wiki/RMI-IIOP), is the best technology for system integration because it is customized for the Java platform. Since 1990 the industry consortium *Object Management Group* (OMG, http://omg.org), including over 200 major companies worldwide, developed the sophisticated distributed object model/technology CORBA and API standards for heterogeneous system integration in various application domains. While CORBA supports more advanced services than web services, it is too aggressive and heavy and thus not successful in business. Since early 2000s the major IT companies, including Microsoft, IBM and Sun (now part of Oracle), acknowledged that no single platform/technology could dominate in all domains, and started to define the common open-specification for a simpler system integration technology based on existing technologies HTTP and XML. Web service's main objectives are simplicity and wide acceptance, not performance. Fundamentally, web services support remote method invocation where the two communicating computers may be separated by the Internet, based on different hardware and software platforms (computer architecture and operating system respectively), and implemented with different programming languages. The direct consumers of web services are programs, not people.

The main strength of web services include

1. It supports method invocation across heterogeneous systems.
2. It is not a proprietary technology and therefore inexpensive to adopt.
3. It requires minimal programming in its adoption.
4. Systems don't need to install special software for consuming different web services. The client system only needs to have the generic web service toolkit and XML library for its specific platform.
5. Applications access the remote web services through port 80 which is reserved for the web and normally not blocked by enterprise firewalls, so web service deployment is very simple.

The major drawbacks of web services include

1. It is a wrapper technology running on top of multiple existing software layers (web, HTTP, XML) therefore it is not as efficient as those technologies customized for specific platforms.
2. Its specification, especially for XML representation of the advanced data structures, is still not completely standardized. As a result in real projects you may find the need to work at lower levels than we describe in this module.

**Question 11:** Are web services a cutting-edge technology?

**Question 12:** Should you always use web services to integrate components of an information system?

**Question 13:** How could you use web services to support fast transformation of a legacy application to provide services over the Internet?

**Question 14:** Why do we say that web services are easy to deploy?

### 1.5 Web Service Security

While many Web services are open to the public, there are many proprietary web services that are intended for targeted clients. Also, it is very important to avoid malicious intruders from compromising the business data behind the web services or the availability of the web services.

There are three major approaches to secure a web service.

1. Use *Secure Socket Layer (SSL)* protocol to provide a secure communication channel between the web service and its client systems. As a result, intruders cannot easily eavesdrop data between the web service and its client systems.
2. Encrypt SOAP data by the client-side's proxy classes, and decrypt SOAP data at the reception end web object. As a result, even if the intruders intercepted the SOAP data, they cannot easily interpret the data.
3. Implement authentication and authorization mechanisms at OS level or application level.

# 2 Lab Objectives

In this lab you will

1. Develop a Java web service for squaring an integer;
2. Develop a Java client application for consuming the Local Java web service;
3. Run web service client and server on a LAN;
4. Develop a Java client application to consume a public remote .NET web service.

# 3 Lab Setup

You will use the *ubuntu10* VM for this module's labs. Depending on the version of the *ubuntu10* VM, you *may* need to complete the following three steps:

1. Start a terminal window with menu item "Applications|Accessories|Terminal". By default the current work folder is "~" ("/home/user"). If not, run "cd".
2. Run command "sudo chmod -R a+rwx ~/tomcat/webapps/axis" in the terminal window so you can modify files in folder "~/tomcat/webapps/axis".
3. Run command "wget http://csis.pace.edu/lixin/download/webServiceLab.7z" to download "webServiceLab.7z" to *Ubuntu10* home folder "/home/user". Right-click on it in a file browser and choose "Extract Here" to extract folder "/home/user/webServiceLab".

If you decide to use your own computer or VM to work on the labs, you need to use Dr. Lixin Tao's "A Tutorial on Setting up Ubuntu Linux Virtual Machines" (http://csis.pace.edu/lixin/ubuntu/LinuxTutorial.pdf) as reference and complete the following steps:

1. Install the latest Java JDK, and add folder "[JDK-install-folder]/bin" (replacing […] with your actual folder path; and replacing "/" with "\" on Windows) on OS PATH so you could run "javac" from any working folder. Open a terminal or command prompt window, type command "javac" in it and make sure you see "javac" usage information.
2. Install the latest Tomcat.

3. Download file "webServiceResources.7z" from http://csis.pace.edu/lixin/download/webServiceResources.7z" to your computer, and unzip it to create folder "webServiceResources".
4. Inside the new folder "webServiceResources", copy (drag-and-drop, use a USB portable disk as a bridge if necessary) folder "webServiceLab" to "C:\" if you use a Windows PC or "~" if you use a Linux PC; copy folder "endorsed" to folder "[JDK-install-folder]/jre/lib" on your PC; and copy file "axis.war" to folder "[tomcat-install-folder]/webapps" (replacing […] with your actual folder path; replacing "/" with "\" on Windows). If Tomcat is not running, run it to observe the automatic creation of a new folder "[tomcat-install-folder]/webapps/axis". Delete file "[tomcat-install-folder]/webapps/axis.war" or move it to another folder (otherwise you cannot modify the contents in the "axis" folder).

# 4 Lab Guide

## 4.1 Development of a Java Web Service for Squaring an Integer

In this lab, you are going to create a new Java web service that receives one integer and returns its squared value. For example, if input is 2, it will return 4.

The implementation is through a Java class source file. For your convenience, the file has been created in "~/webServiceLab/java/server/SquareIntegerServer.java". Its contents are listed below:

```
public  class  SquareIntegerServer {
   public int square(int x) throws Exception {
      int  result = x*x;
      return  result;
   }
}
```

**Program 1 SquareIntegerServer.java**

Class "SquareIntegerServer" has only one method named "square", which accepts one integer-typed parameter "x", and returns an integer. The business logic for this method is to evaluate the square of the value in variable "x". The result is then returned to the method invoker. The second line of the source code contains clause "throws Exception" because we intend to use this class to implement a web service, and there are many reasons for the network interactions between the method implementation and its remote client to go wrong. We need to explicitly declare that networking errors may happen.

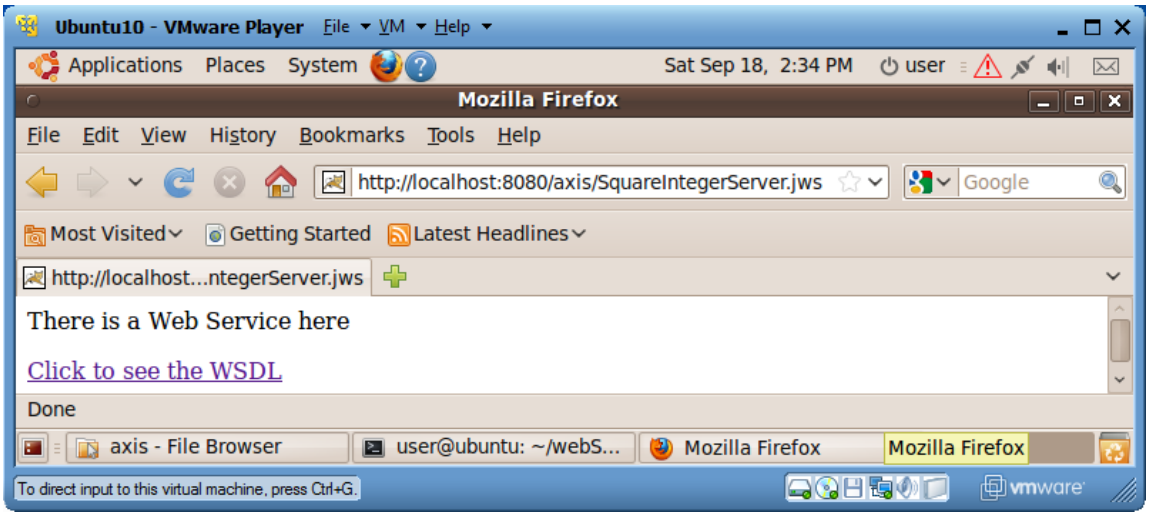
After reviewing the source code, we are ready to deploy it as a web service on our Tomcat web server.

1. Start a terminal window with menu item "Applications|Accessories|Terminal".
2. Run "cd ~webServiceLab/java/server" to change terminal work directory to "~/webServiceLab/java/server".
3. Review the contents of file "SquareIntegerServer.java" with command "gedit SquareIntegerServer.java" Shut down the editor after reviewing.

4. Verify that the source code is a valid Java class by typing the following command in the terminal window to compile the source file:

    javac SquareIntegerServer.java [Enter]

5. In this case you will notice that a new file "SquareIntegerServer.class" is successfully created. In general, if you see error messages during compilation, you need to revise your source code to make it error-free.
6. In a file browser, rename file "SquareIntegerServer.java" to "SquareIntegerServer.jws". Here "jws" stands for "Java web service".
7. In a file browser, (make sure you have run "sudo chmod -R a+rwx ~/tomcat/webapps/axis" beforehand), copy file "SquareIntegerServer.jws" to folder "~/tomcat/webapps/axis", which is the folder for you to deploy any of your Java web services. If this file is already available in folder "axis", replace it with your copy.
8. Since we just deployed a new web component, it is safer to restart Tomcat if it is already running. Run in a terminal window "tomcat-stop" followed by "tomcat-start". Wait until the Tomcat startup is complete. You can now minimize the terminal window, but don't close it.
9. Open a web browser and visit URL http://localhost:8080/axis/SquareIntegerServer.jws. You are going to see a window like the following one:




10. Your new web service is working now! Click on the hyperlink "Click to see the WSDL" to review the WSDL file for this web service. You will notice that the URL address box now contains value "http://localhost:8080/axis/SquareIntegerServer.jws?wsdl". As a matter of fact, given any web service URL, you can always add "?wsdl" to the end of the URL for retrieving the WSDL file for the web service. When you started the Tomcat, the Axis toolkit embedded in the Tomcat compiled our source file "SquareIntegerServer.jws" into a Java servlet with URL http://localhost:8080/axis/SquareIntegerServer.jws, and also generated the web service definition language (WSDL) file for describing the method exposed by this web service. Figure 2 shows the contents of the WSDL file after slight formatting. The lines in bold face specify the format of SOAP response message (for method call return value), SOAP request message (for method invocation information), and the URL for accepting the SOAP requests.

```
<?xml version="1.0" encoding="UTF-8" ?>
 <wsdl:definitions
  targetNamespace="http://localhost:8080/axis/SquareIntegerServer.jws"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://localhost:8080/axis/SquareIntegerServer.jws"
  xmlns:intf="http://localhost:8080/axis/SquareIntegerServer.jws"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 >

 <wsdl:message name="squareResponse">
  <wsdl:part name="squareReturn" type="xsd:int" />
 </wsdl:message>

 <wsdl:message name="squareRequest">
  <wsdl:part name="x" type="xsd:int" />
 </wsdl:message>

 <wsdl:portType name="SquareIntegerServer">
  <wsdl:operation name="square" parameterOrder="x">
   <wsdl:input message="impl:squareRequest" name="squareRequest" />
   <wsdl:output message="impl:squareResponse" name="squareResponse" />
  </wsdl:operation>
 </wsdl:portType>

 <wsdl:binding name="SquareIntegerServerSoapBinding"
                type="impl:SquareIntegerServer">
  <wsdlsoap:binding style="rpc"
                transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="square">
   <wsdlsoap:operation soapAction="" />
   <wsdl:input name="squareRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://DefaultNamespace" use="encoded" />
   </wsdl:input>
   <wsdl:output name="squareResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                namespace="http://localhost:8080/axis/SquareIntegerServer.jws"
                use="encoded" />
   </wsdl:output>
  </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="SquareIntegerServerService">
   <wsdl:port  binding="impl:SquareIntegerServerSoapBinding"
                name="SquareIntegerServer">
    <wsdlsoap:address location="http://localhost:8080/axis/SquareIntegerServer.jws" />
   </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

**Figure 2 WSDL file for the SquareIntegerServer web service**

**Question 15:** Do you write a complete program to implement a web service?

**Question 16:** Is creating a web service a highly technical task?

**Question 17:** What is the relationship between the URL for accepting web service method invocation requests and the URL for displaying the web service WSDL file?

## 4.2 Development of a Java Client Application for Consuming the Local Java Web Service

Now it is time for us to develop a Java program to function as the client of the new web service.

1. In the terminal window, run "cd ~/webServiceLab/java/client" to change work directory to "~/webServiceLab/java/client" .
2. Make sure that your Tomcat is still running (using a web browser to visit http://localhost:8080 and making sure you could see the Tomcat home page). Run the following command on a single line (no line break) in the terminal window to generate source files for your proxy class for your new web service:

```
java  org.apache.axis.wsdl.WSDL2Java
   http://localhost:8080/axis/SquareIntegerServer.jws?wsdl
```

This command will generate four Java source files supporting the web service proxy class. These four Java files all belong to Java package "localhost.axis.SquareIntegerServer_jws", therefore they are located in directory, relative to the current directory "~/webServiceLab/java/client", "localhost/axis/SquareIntegerServer_jws". The name and function of these four Java files are as follows:

a) **SquareIntegerServer.java**
Interface for the web service proxy class to implement

b) **SquareIntegerServerService.java**
Interface for the factory class of the web service proxy objects (proxy objects are not generated by operator **new**, but through method calls to a factory object)

c) **SquareIntegerServerSoapBindingStub.java**
Proxy class source, which implements interface **SquareIntegerServer**

d) **SquareIntegerServerServiceLocator.java**
Factory class of proxy objects; it implements interface **SquareIntegerServerService**

The names of these files and the subdirectories depend on the URL and contents of the WSDL file. When you create proxy classes for a different web service, you need to change the argument to class WSDL2Java to the URL of the WSDL file of that web service. The resulting proxy class files may have different names and package path, but they should follow the same pattern as our example here.

3. Now in directory "C:\webServiceLab\java\client", use a text editor, like *gedit*, to create a Java source file named "SquareIntegerClient.java" with its contents specified below (for your convenience, the file has been created for you; please review its contents):

```
import localhost.axis.SquareIntegerServer_jws.*;

public class SquareIntegerClient {
  public static void main(String[] args) throws Exception {
    int value = 0; // value to be squared
    // The program expects to receive an integer on command-line
    // Program quits if there is no such integer
    if (args.length == 1)  // there is one command-line argument
      value = Integer.parseInt(args[0]);  // parse the string form of integer to an int
    else {
      System.out.println("Usage: java SquareIntegerClient  [integer]");
      System.exit(-1);  // terminate the program
    }
    // Get the proxy factory
    SquareIntegerServerServiceLocator factory =
        new SquareIntegerServerServiceLocator();
    // Generate the web service proxy object
    SquareIntegerServer proxy = factory.getSquareIntegerServer();
    // Access the web service
    int result = proxy.square(value); // invoke server method to square value
    System.out.println("Square of " + value + " is " + result);
  }
}
```

**Program 2 SquareIntegerClient.java**

The file first imports all the Java classes generated by class "WSDL2Java" for supporting the web service proxy object. Then it checks whether there is a command-line argument. If there is no command-line argument, the program prints a usage message and then terminates. Otherwise it parses the string form of the command-line integer into an *int* number, and saves the number in variable *value*. A factory object for generating a proxy object is then created with "new SquareIntegerServerServiceLocator()", and a new proxy object is obtained by calling the factory's method "getSquareIntegerServer()". The actual web service invocation happens on the line with "proxy.square(value)". Even though this method call is to the local proxy object, the proxy's body for method "square" makes a TCP IP connection to the Tomcat servlet specified in the WSDL file, issues an HTTP request carrying a SOAP request message in its *entity body*, receives a response SOAP message from the Tomcat servlet (from the *entity body* of the HTTP response), parses the SOAP response message into a Java *int* value, and returns the value as its own method return value.

4. Now it's time to compile the client program. In the terminal window, type

   **javac SquareIntegerClient.java -source 1.4**

The *javac* command-line switch "-source 1.4" is for informing the Java compiler that this class should be compiled with Java SDK 1.4's API. This is because in Java SDK 1.5 (or 5), "enum" is defined as a Java key word, but they were not in Java SDK 1.4 or earlier. The *Axis* toolkit uses "enum" extensively as a package name, which is not allowed in Java SDK 1.5 (5). You may see two warnings related to this fact, which are harmless. After executing this command, you will notice a new file "SquareIntegerClient.class".

5. Now you can test your web service with your new Java client program. Type the following command, and the second line is the response of your client program. The computation is performed by your web service. You can try to use different integers to replace integer 2.

   **java SquareIntegerClient 2**
   Square of 2 is 4

Congratulations! You have successfully implemented a web service client application in Java. The class name of your client application is not important and you can choose any name meaningful to you.

To consume a web service implemented on another platform, say on Microsoft .NET with C# or Visual Basic .NET, you just follow the same procedure. As long as you provide the right URL for the WSDL file of the web service, the implementation language or platform of the web service is of no relevance to you. This is the real power of web services!

**Question 18:** Is the web service client a person or a program?

**Question 19:** What is the name of the class that you used to create the proxy classes?

**Question 20:** What is the relationship between the folder paths for the proxy classes and the Java package path for the proxy classes?

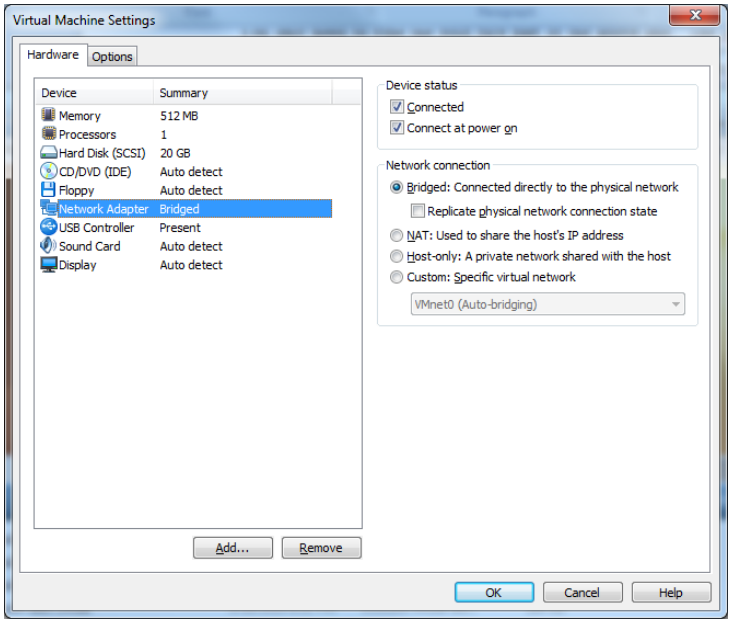
**Question 21:** What is a factory class?

**Question 22:** What are the two most important proxy classes that you use in creating a web service client program?

## 4.3 Running Web Service Client and Server on a LAN

In this lab you learn how to deploy our web service server and client on different Ubuntu10 VMs running on different PCs in a (home) local area network (LAN). We assume that a (*CableVision* or *Verizon*) router runs a Dynamic Host Configuration Protocol (DHCP) server that assigns dynamic IP addresses for your computers in the LAN; and you plan to run the web service on computer A and run the web service client on computer B, both are part of your home LAN. Find out your router's IP address that is used for using a web browser to login to your router and configure it. This is also your DHCP server's IP address. All of your physical or virtual computers must have the same initial three integers in their IP addresses, typically "192.168.0" or "192.168.1", to be able to communicate with each other. As a special case the two computers could be the same one (in this case let the two VM folders differ in folder names, or be located in different file system location). Both of the two computers should have VMware Player installed. Make a copy of VM folder "ubuntu10" of both computers. When you launch each of the VM for the first time, make sure you choose "I copied it" to VMware Player so your VMs would have unique network MAC values.

1. On computer A, set up the square integer web service server on a Ubuntu10 VM as described in Section 4.1.
2. Use VMware Player top menu "VM|Settings …" to launch the "Virtual Machine Settings" window. Choose the "Hardware" tab. Click on the "network Adapter" row. In the right "Network connection" section, check the "Bridged" check box, as shown below:




3. Click on the bottom "OK" button to shut down the "Virtual Machine Settings" window.
4. Use menu item "Applications|Accessories|Terminal" to launch a terminal window, and run in it command "sudo ifconfig". If the "eth0" or "eth1" section of the output message contains an inet address whose first three digits are the same as those of your router's IP address, then your VM's network configuration is complete and this VM can be accessed on your LAN as a physical PC. Otherwise restart your VM.
5. Repeat steps 2-4 on your Ubuntu10 VM on computer B to make it part of your LAN.
6. Set up the "~/webServiceLab" folder on your Ubuntu10 VM on computer B, as explained in Section 4.1.
7. Complete the lab describe in Section 4.2 to develop the web service client, but adjust the URL for the web service WSDL file and the import line of file "SquareIntegerClient.java". In my case my Ubuntu10 VM on computer A has IP address 192.168.0.240. I generated the proxy classes with command

```
java  org.apache.axis.wsdl.WSDL2Java
   http://192.168.0.240:8080/axis/SquareIntegerServer.jws?wsdl
```

Find the path for your new proxy classes, and adjust the import statement of file "SquareIntegerClient.java" accordingly. If the proxy files are in folder "localhost/axis/SquareIntegerServer_jws", then you don't need to change the import statement of file "SquareIntegerClient.java". Sometimes the proxy class package path may match the IP address of the web service server. In my experiment, since my computer A has address 192.168.0.240, my proxy classes are in Java package "_240._0._168._192.axis.SquareIntegerServer_jws". Therefore I need to use text editor "gedit" to change the first line of file "SquareIntegerClient.java" to

```
import _240._0._168._192.axis.SquareIntegerServer_jws.*;
```

8. The web service client should be able to invoke the web service on computer A and provide the same function of squaring integers.

**Question 23:** What is the function of a DHCP server?

**Question 24:** Suppose your home LAN has network mask "255.255.255.0". What if your home computers have IP addresses differ on any of the first three numbers?

**Question 25:** If two VMs are deployed on the same PC and you want them communicate with each other and with the host PC. How should you configure the VMs' network adapters?

**Question 26:** If two VMs are deployed on two different PCs on your LAN and you want them communicate with each other and with the two host PCs. How should we configure the VMs' network adapters?

**Question 27:** On a Linux computer which command will tell us the IP address and network mask of the computer?

## 4.4 Developing a Java Client Application to Consume a Public Remote .NET Web Service

In this lab you will develop a Java client application to consume the remote public web service for finding US holidays in a year. The web service was developed with Microsoft .NET and described at http://www.service-repository.com/service/wsdl?id=98316. The WSDL file for the web service is at http://www.27seconds.com/Holidays/US/Dates/USHolidayDates.asmx?WSDL. That is all information we have about this web service. This lab provides guidance on how to consume unknown public web services.

1. Launch the Ubuntu10 VM.
2. Start a terminal window with menu item "Applications|Accessories|Terminal".
3. Run "cd ~/webServiceLab/java/client" to change the work directory of the terminal window to the folder for developing web service Java clients. The only reason we use this folder is because we have an example web service client program here for our reference. You could conduct this lab in any folder.
6. Run the following command on a single line (no line break) in the terminal window to generate source files for your proxy class for your new web service:

```
java  org.apache.axis.wsdl.WSDL2Java
   http://www.27seconds.com/Holidays/US/Dates/USHolidayDates.asmx?WSDL
```

This command will generate four Java source files supporting the web service proxy class. These files have names "USHolidayDates.java", "USHolidayDatesLocator.java", "USHolidayDatesSoap.java", and "USHolidayDatesSoapStub.java" respectively. These four Java files all belong to Java package "com._27seconds.www.Holidays.US.Dates", therefore they are located in directory, relative to the current directory "~/webServiceLab/java/client", "com/_27seconds/www/Holidays/US/Dates". By comparison with the file names for the Square

Integer proxy class files, we guess class "USHolidayDatesLocator" is the factory for creating the proxy object, and "USHolidayDates" is a java interface listing web service methods that we could call. We use gedit to open file "USHolidayDatesLocator.java" and did find the factory method "USHolidayDatesSoap getUSHolidayDatesSoap()". Therefore our guess about the proxy factory class is confirmed. The return data type "USHolidayDatesSoap" of the factory method told us that the interface for the web service methods is in file "USHolidayDatesSoap.java". Use gedit to review the contents of this file and we see a long list of web service method that we could use including

```
public interface USHolidayDatesSoap extends java.rmi.Remote {

  // Get the date of New Year.
  public java.util.Calendar getNewYear(int getNewYearYear)
    throws java.rmi.RemoteException;

  // Get the date of Martin Luther King Day.
  public java.util.Calendar
    getMartinLutherKingDay(int getMartinLutherKingDayYear)
    throws java.rmi.RemoteException;

  // Get the date of President's Day.
  public java.util.Calendar getPresidentsDay(int getPresidentsDayYear)
    throws java.rmi.RemoteException;
  ……
}
```

All these methods have an integer parameter for the year, and return a Java "java.util.Calendar" object representing a date. You should visit Java API documentation at http://download.oracle.com/javase/7/docs/api/ to learn how to print the dates from the Calendar object. In the center "Packages" column we click on "java.util", then click on the "Calendar" link in the "Class Summary" section of the "Package java.util" page. Here you learn how to retrieve the year, month and day components of a Calendar object.

7. With reference to file "SquareIntegerClient.java", you create the new file "USHolidaysClient.java" with the following contents:

```
import com._27seconds.www.Holidays.US.Dates.*;
import java.util.*;

public class USHolidaysClient {
  public static void main(String[] args) throws Exception {
    int year = 0; // year
    // The program expects to receive an integer (year) on command-line
    // Program quits if there is no such integer
    if (args.length == 1)  // there is one command-line argument
      year = Integer.parseInt(args[0]); // parse the string to an int
    else {
      System.out.println("Usage: java USHolidaysClient  [integer]");
      System.exit(-1);  // terminate the program
    }
    // Get the proxy factory
    USHolidayDatesLocator factory = new USHolidayDatesLocator();
    // Generate the web service proxy object
    USHolidayDatesSoap proxy = factory.getUSHolidayDatesSoap();
    // Access the Web service
    Calendar  cal;
    cal = proxy.getEaster(year);
    cal.add(Calendar.DAY_OF_MONTH, 1);
    System.out.println("Easter: \t" + (cal.get(Calendar.MONTH)+1) +
      "/" + cal.get(Calendar.DAY_OF_MONTH) + "/" + cal.get(Calendar.YEAR));
    cal = proxy.getMothersDay(year);
    cal.add(Calendar.DAY_OF_MONTH, 1);
```

```
        System.out.println("Mother's Day: \t" + (cal.get(Calendar.MONTH)+1) +
          "/" + cal.get(Calendar.DAY_OF_MONTH) + "/" + cal.get(Calendar.YEAR));
        cal = proxy.getFathersDay(year);
        cal.add(Calendar.DAY_OF_MONTH, 1);
        System.out.println("Father's Day: \t" + (cal.get(Calendar.MONTH)+1) +
          "/" + cal.get(Calendar.DAY_OF_MONTH) + "/" + cal.get(Calendar.YEAR));
        cal = proxy.getThanksgivingDay(year);
        cal.add(Calendar.DAY_OF_MONTH, 1);
        System.out.println("Thanksgiving: \t" + (cal.get(Calendar.MONTH)+1) +
          "/" + cal.get(Calendar.DATE) + "/" + cal.get(Calendar.YEAR));
        cal = Calendar.getInstance();
        System.out.println("Today: \t\t" + (cal.get(Calendar.MONTH)+1) +
          "/" + cal.get(Calendar.DAY_OF_MONTH) + "/" + cal.get(Calendar.YEAR));
    }
}
```

Note that the web service methods always return one day earlier (discovery by testing), therefore I added one day to each of the returned dates. Java Calendar uses 0 for January, therefore you need to add 1 to the month value for printing.

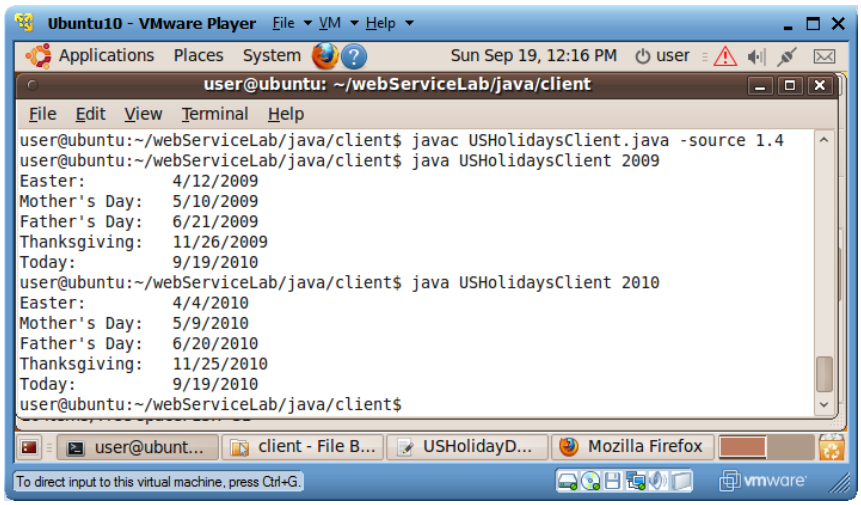
8. To compile the client program, type in the terminal window

   **javac USHolidaysClient.java -source 1.4**

9. To run the client program, type in the terminal window

   **java USHolidaysClient 2010**

   and you can change 2010 to any valid year. The following is the screen capture of one of my test session.




**Question 28:** Do you need know web service server implementation before you could develop a web service client to get its service?

**Question 29:** To use a new web service over the Internet, do you need to install special software, like plug-ins, for that service?

# 5 Review Questions

**Question 30:** There is a public .NET web service with its WSDL file at http://www50.brinkster.com/vbfacileinpt/np.asmx?wsdl that can return a string of all positive prime numbers smaller than or equal to the positive integer that you provide as argument. Develop a Java web application to consume this service.

**Question 31:** Develop a Java web service that supports method "double add(double x, double y)" for returning the summation of two floating point numbers; and develop a Java client program to consume this web service. To convert string "12.5" into floating-point number 12.5, you could use "double d = Double.parseDouble("12.5");".