1 Introduction

Objectives

After studying this chapter, you should be able to:

1. Describe basic elements of a Web Service application
2. Compare and contrast the purposes of Web and Web Service applications
3. Describe the benefits of Web Services
4. Write a simple Web Service application using Java Development Kit (JDK) 6 or later
5. Verify and test a Web Service application

In the early days of the Internet, Web applications delivered static webpages via HTML. Certainly, the development of websites was simpler; however, static content can quickly become outdated; thus, the content management of a website is important.

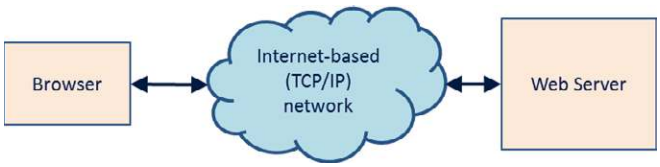

Figure 1-1 *Early Web applications*

In order to provide dynamic content to Web users, 2-tier web applications were realized with the introduction of the Common Gateway Interface (CGI), which retrieves content from external data resources, such as a database. CGI acts as a client in the traditional client-server architecture. A CGI script processes the request and returns the result to the Web server. The server then formats the contents in HTML and returns to the browser for display.

CGI suffered many drawbacks that necessitated changes to the 2-tier architecture. The database was often running on the same machine; therefore, making backups of the data was difficult. CGI was running as a separate process, so it suffered from a context-switching penalty. CGI was not designed for performance, security or scalability.

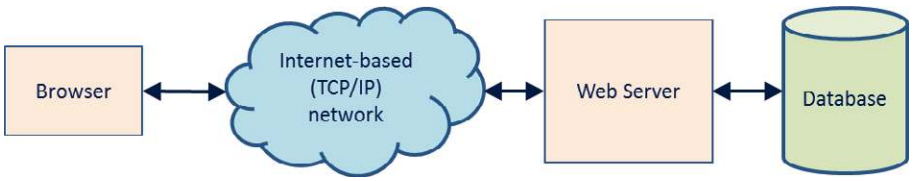

Figure 1-2 *Two-tier web application*

Nowadays, n-tier Web application architecture is commonly used. In this architecture, middleware or an application server is introduced to connect the Web server and the database more efficiently. The performance of an n-tier application is improved because Web servers, middleware and databases can be hosted by separate machines. Each tier can be replicated for the purposes of load balancing. Security is also improved because data is not stored on the Web or application server, which makes it harder for hackers to gain access into the database where data is stored.

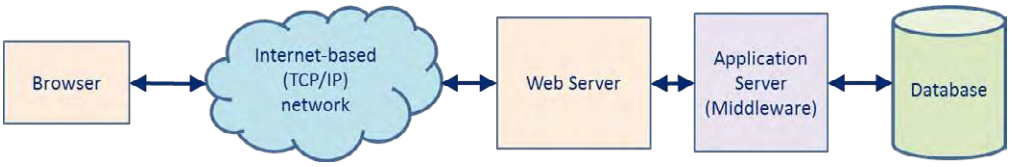

Figure 1-3 *An n-tier web architecture*

A web and an application servers are often run on the same machine; however, it is best practice to run the database server on a separate machine. In a software development environment, all three servers can be hosted on a single machine. In this book, a server is often refered to a software application.

1.1 Browsing the Internet

Before the conception of Web 2.0 (around 1999), the basic use of the World Wide Web (WWW) and the Internet was simple and based on the traditional client-server model with older technologies such as Remote Procedure Call (RPC) or Transaction Processing (TP) Monitors or other middleware that permitted programmable clients.

Consider a typical use case of a person browsing the Internet by means of a browser. The Web server in this example serves dynamic HTML pages using Java Server Pages (JSP) technology. In addition, it uses Enterprise Java Bean (EJB) or Plain-Old-Java-Object (POJO). JSP is oriented toward the delivery of webpages for the presentation layer. EJBs or POJOs are usually used for processing business rules. There are thousands of Web applications that use Java/JEE technology.

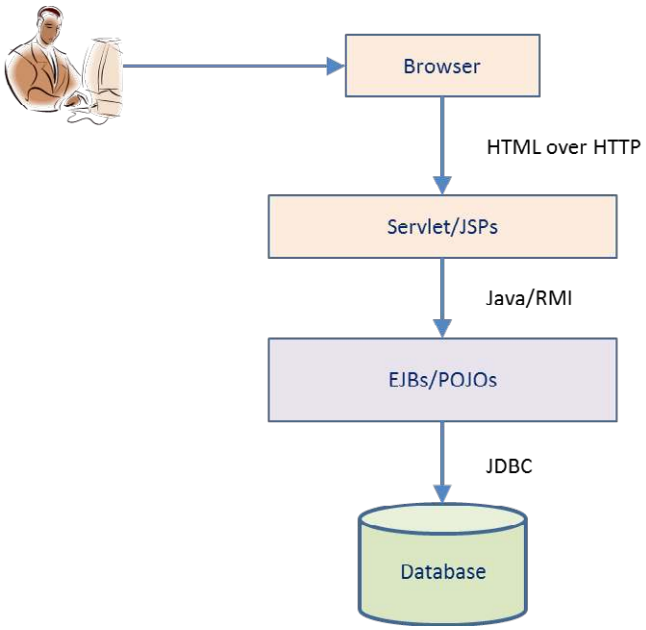

Figure 1-4 *Man-machine interaction*

The Internet architecture was originally designed for human users. HTTP protocol was for exchanging documents (Web or HTML pages). HTML was designed for basic graphical user interface (GUI) applications. Computing resources on a web browser are often idle while the user is browsing the Internet. These available resources prompted the idea of providing more robust web browsing experience. In addition, the idea of business-to-business (B2B) data exchange model also became more feasible. Accordingly, the WS architecture was introduced to support this new type of data exchange.

1.2 Web Service architecture

A service can be one of the three types of interaction: man-to-man, man-to-machine, or machine-to-machine. A restaurant service is an example of man-to-man interaction. A person withdrawing money from an Automated Teller Machine (ATM) is an example of man-to-machine interaction. Machine-to-machine interaction is exemplified by a handheld device, such as a smart device (e.g., a phone or a tablet), synchronizing its address book with Microsoft Outlook. A Web Service is a type of machine-to-machine interaction that uses specific Web standards and technology. A Web Service is a set of programming interfaces, not a set of webpages.

This section begins with a basic definition of a Web Service in order to establish a basic understanding for use in later chapters. More complex aspects of Web Services will be easier to understand when the basic concept of a Web Service is properly explained.

According to W3C website, http://www.w3.org/TR/ws-desc-reqs:

> *A Web Service is a software application identified by a URI whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and [that] supports direct interactions with other software applications using XML based messages via Internet-based protocols.*

A Web Service must involve a Web-based protocol, such as HTTP or Simple Mail Transfer Protocol (SMTP). Other transport protocols may be used, but HTTP is the most common one being used. HTTPS uses Secure Socket Layer (SSL) or Transport Secure Layer (TLS) for secured transport of data. In regard to software development concerns, the difference between HTTP and HTTPS is trivial. HTTP, thus, is used throughout this text.

A Web Service is a software application that requires interaction with another application. WS is a software integration technique for a B2B type of integration. Here, one application acts as a service provider (server) and the others act as service consumers (clients). This is a many-to-one relationship.

'Interface' is defined as "[The] point of interaction or communication between a computer and any other entity" (http://www.thefreedictionary.com). An interface can also be described as an "abstraction of a service that only defines the operations supported by the service (publicly accessible variable, procedures, or methods), but not their implementation" (Szyperski, 2002). For example, in Java, an interface can be defined and then implemented by a concrete class.

Web Service Description Language (WSDL) specifies the service interface and the rules for binding the service consumer and the provider. According to the specification of WSDL 1.1, WSDL is defined as "an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information" (http://www.w3.org/TR/wsdl). WSDL defines how a consumer can interact with a service via a concrete network protocol and message format using eXtended Markup Language (XML).

XML is a profile (subset) of Standard Generalized Markup Language (SGML). SGML is a metalanguage, i.e., a language that describes other languages. Unlike HyperText Markup Language (HTML), which is used to serve static webpages, XML allows the author to create his or her own tags. Thus, XML facilitates the data and document processing functions.

Web Service relies on Simple Object Application Protocol (SOAP) as its transport. As its name implies, SOAP is a lightweight protocol that can be used to exchange structured messages (i.e., XML). SOAP 1.2 is the latest version. WSDL 1.1 supports SOAP 1.1, HTTP GET/POST, and MIME.

A service can be defined, published and discovered using some type of service registry. Current supporting service registries include electronic business XML (ebXML), Universal Discovery, Description and Integration (UDDI), and Metadata Registry (MDR). UDDI is usually a good idea; however, it is not widely used except in a private network of services.

RPC is a powerful technique that provides distributed computing capabilities across a network of machines. RPC is a form of interprocess communication that enables function calls between applications that are located across different (or the same) locations over a network. It is best suited for client-server programming.

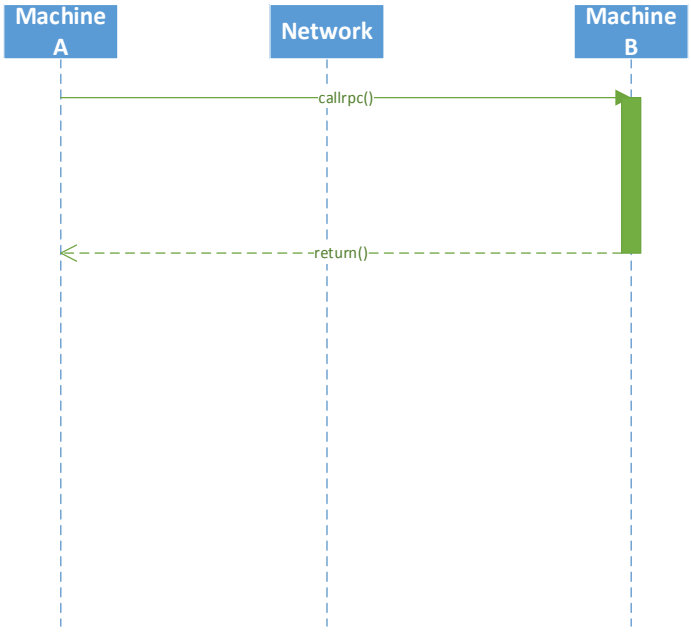

Figure 1-5 *Remote Procedure Call (RPC)*

Web Services can be used to help solve several problems in Enterprise Application Integration (EAI). Integrating existing applications for a business solution is a complex and time-consuming task. Applications that were written in different computer languages, such as C/C++, JAVA, Visual Basic, and FORTRAN, have unique logical interfaces to the external world, which makes the integration of these applications difficult, complex and time-consuming. Applications that are running on different machine architectures, such as SUN, Personal Computer, IBM Mainframes, IBM A/S 400, have unique physical interfaces to the external world. Integrating these applications is also challenging. Applications running on machines that are interconnected through a network are also difficult to integrate. The challenges of EAI arise in three main areas:

- Language barriers – XML is a standardized language that is used for message exchange
- Platform barriers – SOAP has been implemented on many platforms (e.g., Unix, Windows)
- Network barriers – HTTP and SMTP are standardized network protocols

WS can serve as an enabling technology for application integration. WS, as mentioned earlier, places thes following major standards in focus: XML, SOAP, WSDL, UDDI and HTTP.

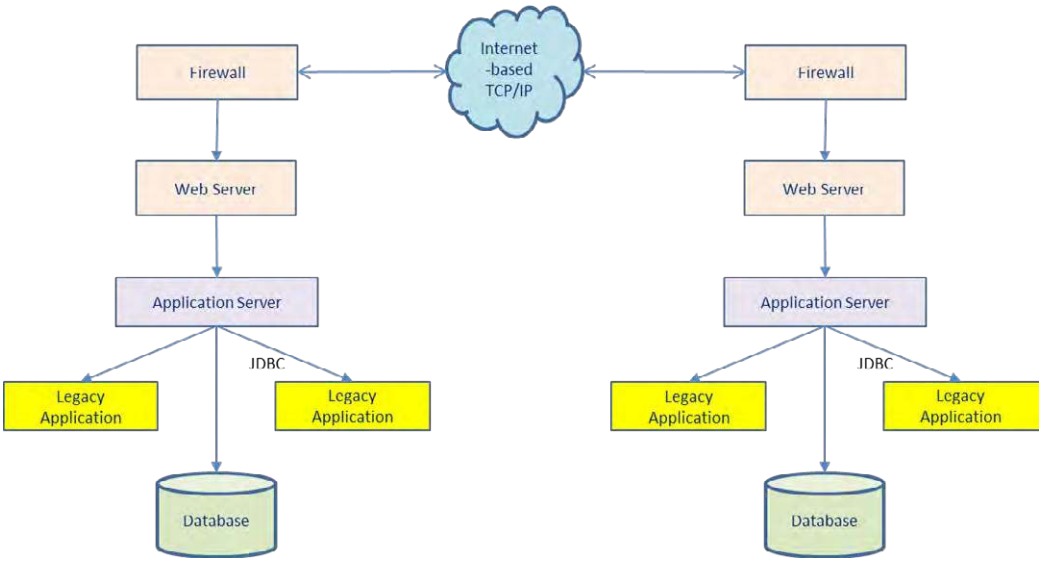

Figure 1-6 *Business-to-Business integration*

SOAP sequence diagram

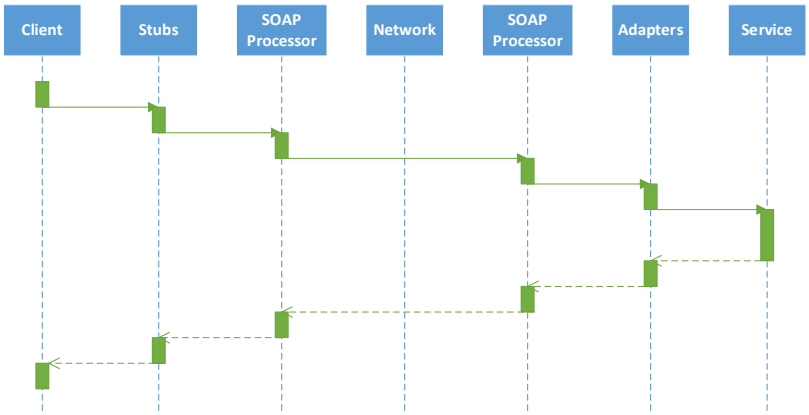

Figure 1-7 *Sequence diagram of SOAP*

Service requester – the client that consumes or requests the service
Service provider – the entity that implements the service and fulfill the service requests
Service registry – a listing like a phonebook where available services are listed and described in full

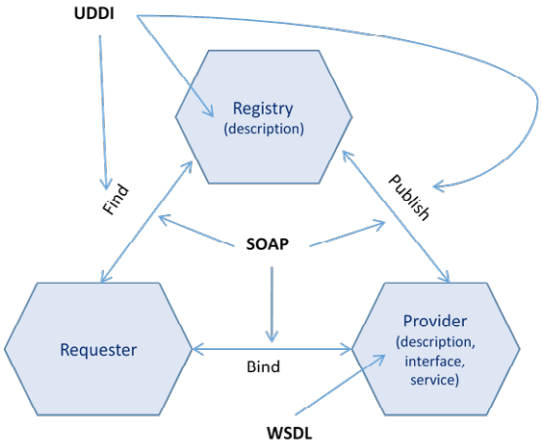

Figure 1-8 *Web Service Architecture*

1.3 Benefits of Web Services

Web Services provide many benefits:

1. Platform-independent: Web Services are now available in nearly all platforms:
   a) Hardware: mainframe, midrange, personal and mobile devices
   b) Operating systems: UNIX, Windows, Mainframe OS, Android, and iPhones
2. Reuse of existing networking infrastructure: HTTP, SMTP, and JMS protocols
3. Loose-coupling of software components promotes software reuse
4. Reduced integration cost and increased integration speed
5. Open architecture and communication protocols

1.4 Program a HelloWorld Web Service

The concept of a Web Service can be difficult to comprehend without seeing a concrete example of how a Web Service is created and used. The top-down approach starts with a WSDL file that describes the services. The top-down approach may increase the level of interoperability and allow more control of the WS, wehereas the bottom-up approach starts at the low level of the Java bean or enterprise Java bean (EJB) and is faster and easier.

The following steps can be used to create and test a simple WS application:

1. Run Eclipse IDE, create a new Java project, and name it 'java-ws'.
2. Run Server.java as a Java program.
3. Verify the WSDL and the associated schema for the service endpoint: http://localhost:9999/HelloWorld?wsdl.
4. Use SOAPUI software to test the HelloWorld Web Service.
5. Create Java Web Service client code.

1.4.1 Create a Project

In the example above, the bottom-up approach is used. This example requires Java 6 or later. A Web Service called 'Hello World' is created with the method called 'say', which requires one String parameter. To create a Java project under Eclipse IDE, perform the following steps:

1. Run Eclipse IDE.
2. Choose File → New → Java Project. Use all default and name it 'java-ws'.
3. Expand the java-ws project, then right-click on the src directory and choose New → Package. Name the package 'com.bemach.ws.hello'.
4. Similarly, create another Java package com.bemach.ws.server.
5. Create two Java classes – com.bemach.ws.hello.HelloWorld.java and com.bemach.ws.server.Server.java.

1.4.2 Create a Web Service

A classic HelloWorld class of Java can be written in a few lines of code. The purpose is to make sure that a Java Virtual Machine is properly installed and ready for programming.

Listing 1-1. HelloWorld.java

```
package com.bemach.ws.hello;
/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

public class HelloWorld {
   public String say (String name) {
      return String.format("Hello, %s!", name);
   }

   public static void main (String[] args) {
      String msg = new HelloWorld().say("Johnny, B. Good");
      System.out.println(msg);
   }
}
```

Output:

```
Hello, Johnny B. Good!
```

In this way, the HelloWorld program is transformed into a WS application. This is a basic WS application using the reference implementation of JAX-WS by the Java language.

To transform the HelloWorld program from a simple Java bean into a Java WS application, four WS annotations – namely, @WebService, @SOAPBinding, @WebMethod and @WebParam – are decorated as follows:

Listing 1-2. HelloWorld.java with Web Service Annotations

```
package com.bemach.ws.hello;
/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

import java.util.logging.Logger;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;

@WebService
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT)
public class HelloWorld {
   private static final Logger LOG = Logger.getLogger(HelloWorld.class.getName());

   @WebMethod
   public String say (@WebParam(name="name") String name) {
      LOG.info("Web service is called!");
      return String.format("Hello, %s!", name);
   }

   public static void main (String[] args) {
      String msg = new HelloWorld().say("Johnny, B. Good");
      LOG.info(msg);
   }
}
```

Annotations indirectly affect the sematics of the program via tools and libraries. The @WebService annotation indicates that the class will implement a WS. The @SOAPBinding annotation indicates the style of the SOAP to be used. In this example, the style is DOCUMENT as opposed to RPC. @WebMethod indicates an operation of the WS to be created. Lastly, the @WebParam indicates how the parameter is named inside the WSDL.

1.4.3 Create a HTTP Server

To host the service endpoint, a WS requires an HTTP server. An Apache JEE Tomcat server can be used; however, a basic server can be created using only Java. In this example, we created an Endpoint with a specific URL that ties to an implementation of the WS. In this case, the URL is http://localhost:9999/java-ws/hello and the implementation is the HelloWorld object svc.

Listing 1-3. Server.java class

```
package com.bemach.ws.server;

/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

import java.util.logging.Logger;

import javax.xml.ws.Endpoint;
import javax.xml.ws.EndpointReference;

import com.bemach.data.DbConfig;
import com.bemach.ws.doc.employees.EmployeeDocData;
import com.bemach.ws.hello.HelloWorld;
import com.bemach.ws.rpc.employees.EmployeeRpcData;

/**
 *
 */
public final class Server {
   private static final Logger LOG = Logger.getLogger(Server.class.getName());
   private static final String MYSQL_DRIVER="com.mysql.jdbc.Driver";
   private static final String DB_HOST = "saintmonica";
   private static final String DB_PORT = "3306";
   private static final String DB_SID = "employees";
   private static final String DB_USER = "empl_1";
   private static final String DB_PSW = "password";
   private Server() {

   }

   protected static DbConfig getDbConfig() {
      DbConfig dbCfg = new DbConfig();
      dbCfg.setDriverName(MYSQL_DRIVER);
      dbCfg.setHost(DB_HOST);
      dbCfg.setPort(DB_PORT);
      dbCfg.setDb(DB_SID);
      dbCfg.setUid(DB_USER);
      dbCfg.setPsw(DB_PSW);
      return dbCfg;
   }

   private static final String HOST_NAME = "localhost";
   private static final String PORT_NO = "9999";
   private static final String HELLO_SVC_NAME = "java-ws/hello";
   private static final String RPC_EMPL_SVC_NAME = "rpc/employees";
   private static final String DOC_EMPL_SVC_NAME = "doc/employees";
   private static final String PROTOCOL = "http";
```

```
protected static SvrConfig getSvrConfig() {
   SvrConfig svrCfg = new SvrConfig();
   svrCfg.setListenHostname(HOST_NAME);
   svrCfg.setListenPort(PORT_NO);
   svrCfg.setListenInterface(HELLO_SVC_NAME);
   svrCfg.setListenProtocol(PROTOCOL);
   return svrCfg;
}

protected static Endpoint publish(SvrConfig cfg, Object svc) {
   String url = String.format("%s://%s:%s/%s",
            cfg.getListenProtocol(),
            cfg.getListenHostname(),
            cfg.getListenPort(),
            cfg.getListenInterface());
      Endpoint ep = Endpoint.publish(url, svc);
      EndpointReference epr = ep.getEndpointReference();
      LOG.info("\nEndpoint Ref:\n"+epr.toString());
      return ep;
}

protected static void startHelloWorld() {
   SvrConfig cfg = getSvrConfig();
   cfg.setListenHostname(HOST_NAME);
   cfg.setListenInterface(HELLO_SVC_NAME);
   cfg.setListenPort(PORT_NO);
   cfg.setListenProtocol(PROTOCOL);

   HelloWorld hello = new HelloWorld();
      publish(cfg, hello);
      LOG.info("HelloWorld service started successfully ...");
}

protected static void startRpcEmployees() {
   SvrConfig svrCfg = getSvrConfig();
   svrCfg.setListenHostname(HOST_NAME);
   svrCfg.setListenInterface(RPC_EMPL_SVC_NAME);
   svrCfg.setListenPort(PORT_NO);
   svrCfg.setListenProtocol(PROTOCOL);
      DbConfig dbCfg = getDbConfig();
      svrCfg.setDbCfg(dbCfg);

   EmployeeRpcData rpcEmpl = new EmployeeRpcData(dbCfg);
      publish(svrCfg, rpcEmpl);
      LOG.info("Employees (RPC) service started successfully ...");

}
```

```
    protected static void startDocEmployees() {
        SvrConfig svrCfg = getSvrConfig();
        svrCfg.setListenHostname(HOST_NAME);
        svrCfg.setListenInterface(DOC_EMPL_SVC_NAME);
        svrCfg.setListenPort(PORT_NO);
        svrCfg.setListenProtocol(PROTOCOL);
            DbConfig dbCfg = getDbConfig();
            svrCfg.setDbCfg(dbCfg);

        EmployeeDocData docEmpl = new EmployeeDocData(dbCfg);
            publish(svrCfg, docEmpl);

            LOG.info("Employees (Document) service started successfully ...");
    }

    /**
     * Start WS Server with multiple service endpoints...
     *
     * @param args
     */
    public static void main(String[] args) {
        startHelloWorld();
        startRpcEmployees();
        startDocEmployees();
  }
}
```

When the Server program runs, it calls the startHelloWorld method to create a WS implementation that ties with a unique URL. The endpoint is then published and ready for receiving requests from a remote client. For this simple program, Ctrl-C can be used to stop the server.

The DbConfig.java class is a simple placeholder for the required parameters for database access and for the URL. In a later example, Java code is used to implement data access to the database. Remember, though, that this is sample code; therefore, the password is displayed or stored in the clear. In a business or secure environment, passwords are entered each time or stored encrypted.

1.5 Host a Web Service

In Eclipse IDE, perform the following actions:

1. To run, open Server.java class. Choose Run → Run As → Java Application.

An Eclipse project would look like this:

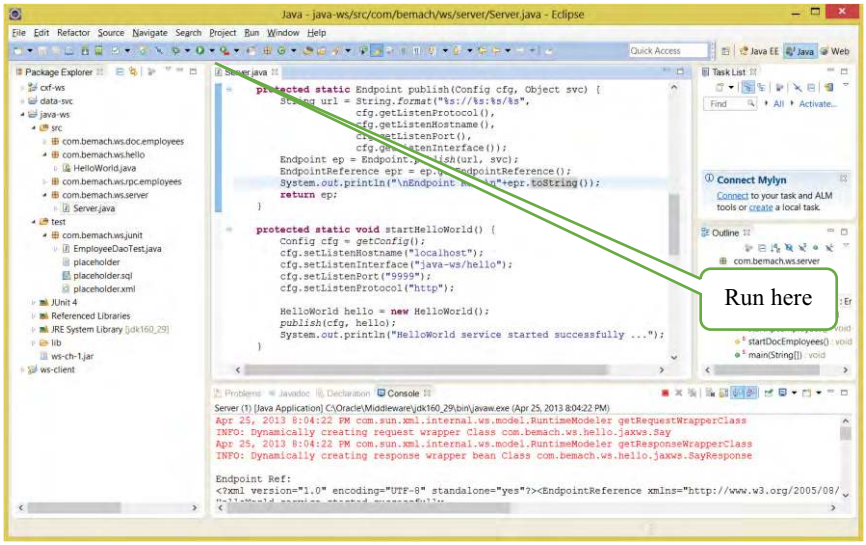

Figure 1-9 *An Eclipse Java project for the HelloWorld Web Service*

1.6 Verify a Web Service

View the HelloWorld's WSDL:

Open a browser and go to this URL: http://localhost:9999/java-ws/hello?WSDL

Listing 1-4. HelloWorld WSDL

```
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://hello.ws.bemach.com/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://hello.ws.bemach.com/"
   name="HelloWorldService">
   <types>
      <xsd:schema>
         <xsd:import namespace="http://hello.ws.bemach.com/"
            schemaLocation="http://localhost:9999/java-ws/hello?xsd=1" />
      </xsd:schema>
   </types>
   <message name="say">
      <part name="parameters" element="tns:say" />
   </message>
   <message name="sayResponse">
      <part name="parameters" element="tns:sayResponse" />
   </message>
   <portType name="HelloWorld">
      <operation name="say">
         <input message="tns:say" />
         <output message="tns:sayResponse" />
      </operation>
   </portType>
   <binding name="HelloWorldPortBinding" type="tns:HelloWorld">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
         style="document" />
      <operation name="say">
         <soap:operation soapAction="" />
         <input>
            <soap:body use="literal" />
         </input>
         <output>
            <soap:body use="literal" />
         </output>
      </operation>
   </binding>
   <service name="HelloWorldService">
      <port name="HelloWorldPort" binding="tns:HelloWorldPortBinding">
         <soap:address location="http://localhost:9999/java-ws/hello" />
      </port>
   </service>
</definitions>
```

Encoding style

Service endpoint

Figure 1-10. *The WSDL of the HelloWorld Web Service*

To view the associated XML schema, go to this URL: http://localhost:9999/java-ws/hello?xsd=1

Listing 1-5. HelloWorld XSD

```
<xs:schema xmlns:tns="http://hello.ws.bemach.com/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
   version="1.0" targetNamespace="http://hello.ws.bemach.com/">
   <xs:element name="say" type="tns:say" />
   <xs:element name="sayResponse" type="tns:sayResponse" />
   <xs:complexType name="say">
      <xs:sequence>
         <xs:element name="name" type="xs:string" minOccurs="0" />
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="sayResponse">
      <xs:sequence>
         <xs:element name="return" type="xs:string" minOccurs="0" />
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

Figure 1-11. *The XML schema associated with the HelloWorld Web Service.*

1.7 Test a Web Service with SOAPUI

SOAPUI is a software that enables software developers and integrators to test Web Services. Similar to Eclipse IDE, SOAPUI is a project-based application.

1. Run SOAPUI program.
2. Select File → New SOAPUI project.
3. Fill in the Project Name and the Initial WSDL/WADL.

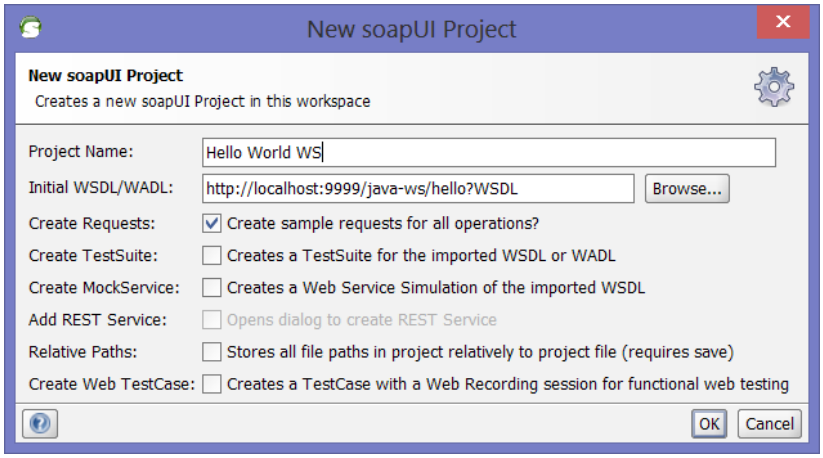

Figure 1-12 *Create a SOAPUI project for the HelloWorld Web Service*

To execute a SOAP operation, take the following steps:

1. On the left panel, double-click on Request 1.
2. Fill in the blank between <arg0> and </arg0>.
3. Click on the green triangle on the top left panel of the request.
4. View the SOAP response on the right panel.

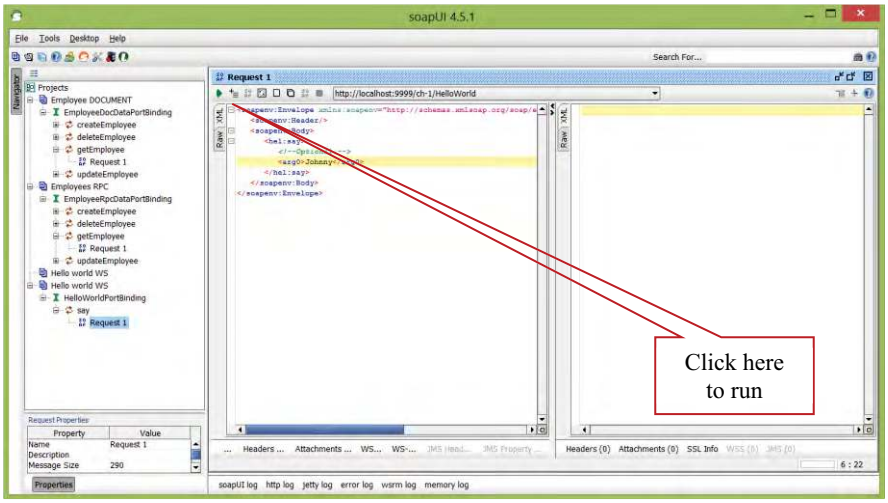

Figure 1-13 *Opening the HelloWorld WSDL*

To troubleshoot at the HTTP layer, click on the 'http log' button on the bottom of screen.

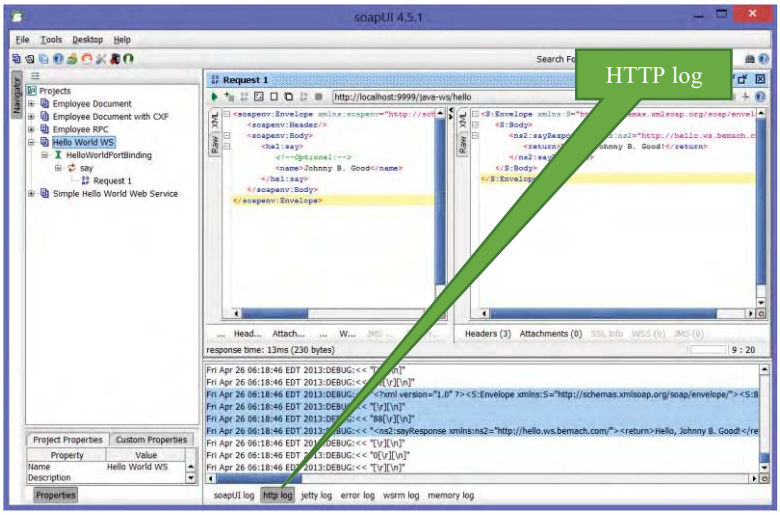

Figure 1-14 *Call an operation (method) of a Web Service*

1.7.3.1 SOAP Request:

The SOAP processor generates this request and sends it across the network to a WS invoking an operation *say* with a simple String argument.

Listing 1-6. A SOAP Request Message

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:hel="http://hello.ws.bemach.com/">
   <soapenv:Header />
   <soapenv:Body>
      <hel:say>
         <name>Johnny B. Good</name>
      </hel:say>
   </soapenv:Body>
</soapenv:Envelope>
```

1.7.3.2 SOAP response:

A SOAP response shows a simple return of a message string.

Listing 17. A SOAP Response Message

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:sayResponse xmlns:ns2="http://hello.ws.bemach.com/">
         <return>Hello, Johnny B. Good!</return>
      </ns2:sayResponse>
   </S:Body>
</S:Envelope>
```

1.8 Create a Web Service Client

A WS client code is simple to write; however, the amount of code required behind the scenes in order to ease the amount of coding on the client side can be substantial. Generated code enables a client application to call WS operations as it would normally do with another Java bean. This makes the programming of a client WS application a bit simpler. In the next chapter, we will use SAAJ APIs to create a Java client code that calls the HelloWorld Web Service.

The process of creating a Java Web Service client to call the HelloWorld Web Service involves the following steps:

1. Create a java-ws-client project in Eclipse.
2. Generate WS client stub from a service endpoint (http://localhost:9999/java-ws/hello?WSDL).
3. Write a Java client class.

1.8.1 Create a Project

1.8.2 Generate Web Service Stub

First, a generated WS client code is generated using a readily available tool, `wsimport`, from the Java JDK package. Second, a client code is written using the generated code.

1. Open a command prompt or a Unix terminal.
2. Go to the java-ws-client project directory.
3. Create a 'generated' directory.
4. Create a 'lib' directory.
5. Go to the 'generated' directory.
6. Run the following command:

```
wsimport -d . http://localhost:9999/java-ws/hello?wsdl
```

7. Package the generated client code:

```
jar cvf ../java-ws-generated.jar *
```

8. Move the java-ws-generated.jar file to the 'lib' directory.

1.8.3 Create Web Service Client

9. Return to Eclipse and refresh the Java project:
   a) Choose java-ws project.
   b) Choose File → Refresh.
   c) From project properties, choose Java Build Path/Libraries.
   d) Click on Add JARs and add the java-ws-generated.jar file.
   e) Click OK.

10. Create a new Java package: com.bemach.ws.hello.client.
11. Create a new Java class: HelloWorldClient.java.

Listing 1-8. A HelloWorld Web Service Client

```
package com.bemach.ws.hello.client;
/**
 * 2013 (C) BEM, Inc., Fairfax, Virginia
 *
 * Unless required by applicable law or agreed to in writing,
 * software distributed is distributed on an
 * "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
 * KIND, either express or implied.
 *
 */

import java.net.MalformedURLException;
import java.net.URL;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.xml.namespace.QName;
import javax.xml.ws.Service;

import com.bemach.ws.hello.HelloWorld;
import com.bemach.ws.hello.HelloWorldService;

/**
 * The following code is a normal way of going about to call
 * a web services using Java code.
 * It is much easier to comprehend.
 *
 */
public class HelloWorldWSClient {
   private static final Logger LOG = Logger.getLogger(HelloWorldWSClient.
class.getName());

   public static void main(String[] args) {
      HelloWorldWSClient client = new HelloWorldWSClient();
      try {
         client.say("Johnny B. Good");
      } catch (Exception e) {
         LOG.log(Level.SEVERE,"ERROR:"+e);
      }
   }

   public void say (String name) throws MalformedURLException {
      LOG.info("service ... ");

      QName qName = new QName("http://hello.ws.bemach.com/", "HelloWorldService");
      URL url = new URL("http://localhost:9999/java-ws/hello");
      Service service = HelloWorldService.create(url, qName);
      HelloWorld port = (HelloWorld)service.getPort(HelloWorld.class);

      String returnMsg = port.say(name);
      LOG.info("return: "+returnMsg);
   }
}
```

1.9 Run a Web Service Client

Output

```
Web Service response: Hello, Johnny B. Good!
```

1.10 References

Gottschalk, K., Graham, S., Kreger, H., & Snell, J. (2002). Introduction to Web Services architecture. *IBM Systems Journal, 41*(2), 170–177.

Kleijnen, S., & Raju, S. (2003). An Open Web Services Architecture. *Queue, 1*(1), 38–46.

Martin, J., Arsanjani, A., Tarr, P., & Hailpern, B. (2003). Web Services: Promises and Compromises. *Queue, 1*(1), 48–58.