Network Computing Lecture 16

Professor Louise E. Moser

Spring 2012

Web Services

- A **Web Service** is a service offered by one application to other applications on the World Wide Web
- More specifically, a Web Service is a **server-side application** that implements procedures that clients can call
- Moreover, a Web Service is deployed in a **server-side container** that is either
  - A **Servlet container** in the Apache Tomcat servlet engine, or
  - An **EJB container** in an EBJ/J2EE application server such as JBoss

Web Services

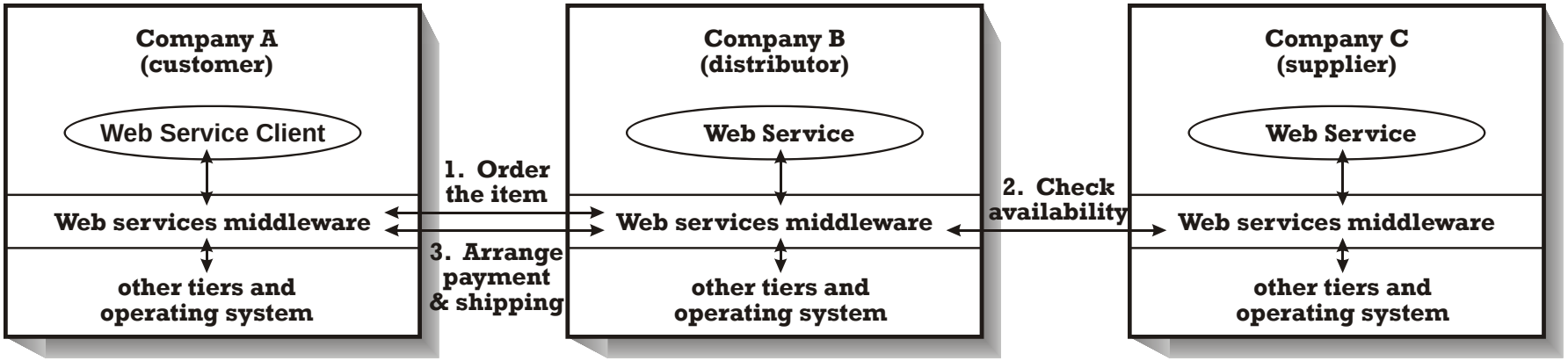
- In a typical Web Services scenario
  - An application sends a request to a Web Service at a given URL using the **SOAP** protocol over HTTP
  - The Web Service receives the request, processes it, and returns a response
- Web Services and their consumers are often businesses, in which case Web Services are **Business-to-Business (B2B)** applications
  - A business can be both a consumer and a provider of Web Services
- The most important requirement of a Web Service is that it is **interoperable** across clients and servers

Web Services Examples

- A **Stock Quote Service** where
  - The request from the client asks for the current price of a specific stock
  - The response of the Web Service is the price of that stock
- A **Parcel Delivery Service** where
  - The request contains one or more delivery destinations
  - The response of the Web Service is the most cost-effective delivery route
    - The time it takes to calculate the best route depends on the complexity of the route, so the response is sent separately from the request, i.e., asynchronously
- A **Wholesale Coffee Bean Distributor** that is
  - A consumer when it uses a Web Service to check on the availability of coffee beans
  - A provider when it supplies a prospective customer with different suppliers' prices for coffee beans

Web Services in B2B Applications

- Company A (the customer) orders goods from Company B (the distributor)
- Company B checks the availability of the goods from Company C (the supplier) and arranges payment and shipping of the goods to Company A, without human intervention



Web Services Technology

- **Web Services** are based on the following standards
  - **eXtensible Markup Language (XML)** - Defines the syntax of Web Services documents so that the information in those documents is self-describing
  - **Simple Object Access Protocol (SOAP)** - Provides a standard way to encode different protocols and interaction mechanisms in XML, so that XML documents can be easily exchanged across the Internet
  - **Web Services Description Language (WSDL)** - An advanced form of Interface Definition Language (IDL) that has been augmented to define additional aspects of a service description
  - **Universal Description Discovery and Integration (UDDI)** - Used by the Service Registry where service providers publish and advertise available services, and service consumers query and search for services

eXtensible Markup Language

- The **eXtensible Markup Language (XML)** makes **data portable** and allows data to be integrated
  - Internally, **within an enterprise** for sharing legacy data between different departments of the enterprise
  - Externally, **between enterprises** for sharing data
- In contrast, **Java** makes **code portable** and allows enterprises using different computing platforms to communicate with each other

Simple Object Access Protocol

- The **Simple Object Access Protocol (SOAP)** defines standards for **XML messaging** and **mapping of data types**, so that applications adhering to those standards can communicate with each other
- A **JAX-RPC Remote Procedure Call** is implemented as a request-response SOAP message
  - A client written in a language other than Java can access a Web Service developed and deployed on the Java platform
  - Conversely, a client written in Java can communicate with a Web Service developed and deployed on another platform

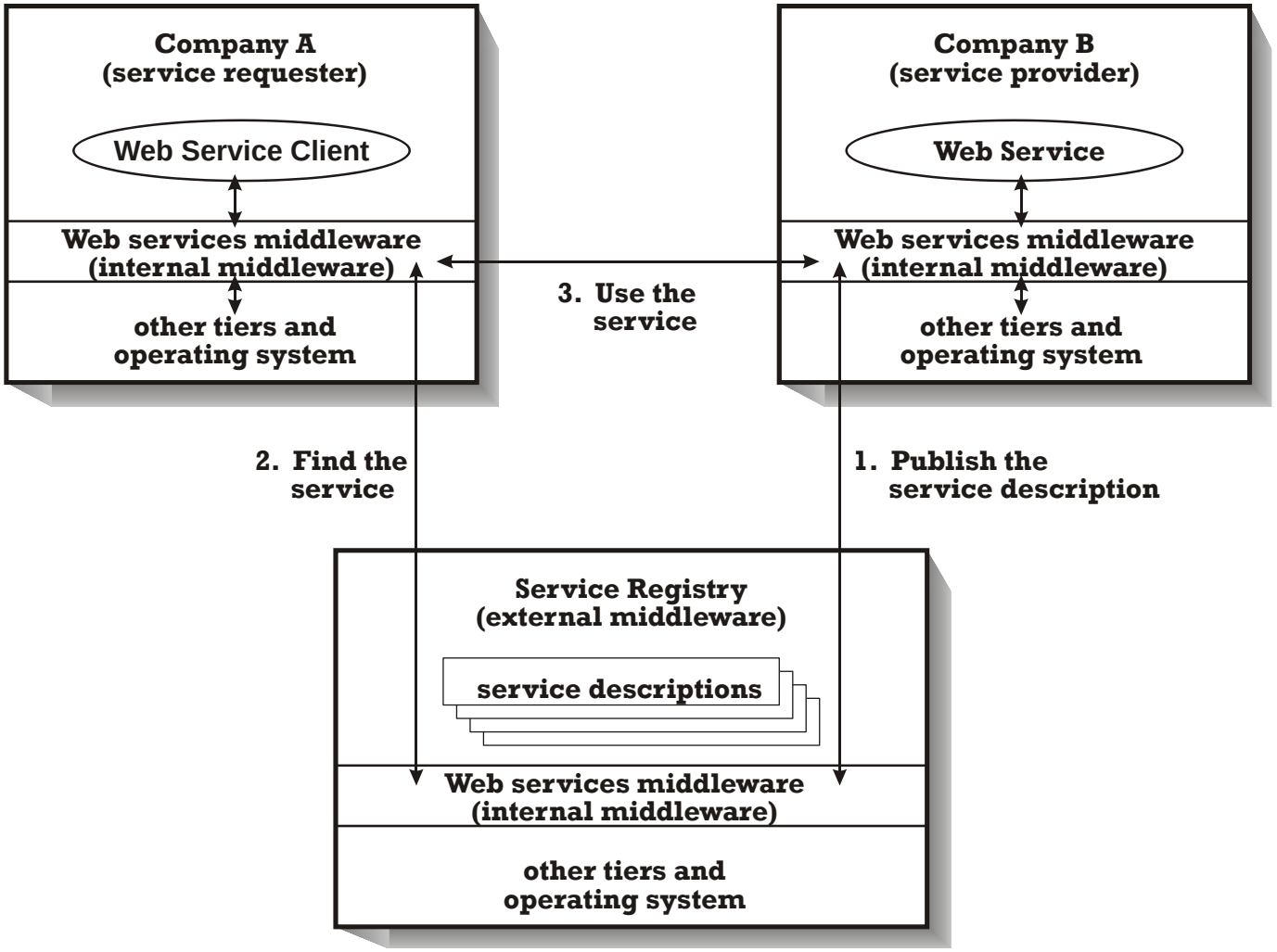
Web Services Description Language

- A Web Service can make itself available to potential clients via a **Web Services Description Language (WSDL)** document
  - A **WSDL document** is an XML document that gives information about a Web Service (its name, operations that can be called on it, parameters for those operations, the location to send requests)
  - A client uses the WSDL document to discover what the Web Service offers and how to access it
- WSDL allows you to describe a Web Service as an **XML document**, which makes the description **portable**

Service Registry

- An **XML business registry** provides WSDL documents that contain information about Web Services
- There are two standards for business registries
  - **ebXML Registry and Repository standard**
  - **Universal Description Discovery and Integration (UDDI) project**
- When searching an XML business registry, a client might get several WSDL documents, one for each Web Service that satisfies its search criteria

Service Registry Usage



A Java Web Service

- A **Java Web Service** basically consists of two files that contain
  - An **interface** that declares the Web Service's remote methods
  - A **class** that implements those methods
- The interface extends *java.rmi.Remote* and its methods throw a *java.rmi.RemoteException*

Java APIs for XML

- The **Java Web Services Developer Pack (WSDP)** supports Java APIs for XML that make it easier to use XML from Java application programs
  - The Java WSDP includes JAR files implementing the APIs, as well as documentation and examples
  - The examples in the Java WSDP run in a Servlet container in Apache Tomcat and also in an EJB container in an EJB/J2EE application server, where the Java WSDP JAR files are installed
- The Java APIs for XML fall into two categories
  - **Document-oriented**
  - **Procedure-oriented**

Java APIs for XML

- **Document-Oriented**
  - **Java API for XML Processing (JAXP)** - Processes XML documents using various parsers
  - **Java Architecture for XML Binding (JAXB)** - Processes XML documents using schema-derived Java Beans classes
  - **SOAP with Attachments API for Java (SAAJ)** - Sends SOAP messages over the Internet in a standard way
- **Procedure-Oriented**
  - **Java API for XML Registries (JAXR)** - Provides access to business registries and allows sharing of information
  - **Java API for XML-based RPC (JAX-RPC)** - Sends SOAP procedure calls to remote computers across the Internet and receives the results of the calls

Simple Object Access Protocol

- The **Simple Object Access Protocol (SOAP)** defines a standard for the exchange of XML documents
- The SOAP specification available at *http://www.w3.org/* defines
  - What is required and optional in a SOAP message
  - How data can be encoded and transmitted
- SOAP supports both JAX-RPC and SAAJ
  - SAAJ is the lower-level alternative to JAX-RPC for SOAP messaging
- We will learn more about SOAP in the next lecture

JAX-RPC

- **JAX-RPC** is the Java API that allows you to develop and use Web Services
- JAX-RPC is based on **Remote Procedure Call (RPC)**
  - A client of a Web Service makes a Java method call
    - All the internal marshalling, unmarshalling, and transmission are done automatically under the covers
  - On the server side, the Web Service simply implements the application-level service that it offers
    - It does not need to bother with the underlying implementation mechanisms
- JAX-RPC typically uses the **two-way request/response synchronous (blocking) model**
  - However, it can also use **asynchronous communication** and **one-way messaging** to send documents or document fragments

Stubs and Ties

- A tool named **wscompile** uses a WSDL document to create **stubs**, the low-level classes that a **client** needs to be able to communicate with a remote Web Service
- Another tool named **wsdeploy** creates **ties** (skeletons), the low-level classes that the **Web Service** needs to communicate with a remote client, and that can also be used to create WSDL documents
- The JAX-RPC mechanism uses the stubs and ties under the covers
  - On the client side, it converts the client's remote method call into a SOAP message and sends it to the Web Service as an HTTP request
  - On the server side, the JAX-RPC runtime system receives the client's request, translates the SOAP message into a method call, and invokes the method
  - Then, the Web Service processes the request
  - The JAX-RPC runtime system goes through similar steps in reverse to return the response to the client
- These mechanisms are invisible to the clients and the Web Services

Creating a Java Web Service

- First, you **write the Web Service's interface and implementation class**
- Next, you **run the mapping tools**, which **generate the stub and tie classes** from the interface and its implementation, and also other classes as needed
- You can also **use the mapping tools to create the WSDL description** for the Web Service
- Next, you **package the Web Service in a Web Application Archive (WAR) file**, to make it easy to distribute and install
- Finally, you **deploy the Web Service**

Example: Coffee Order Interface

- A wholesale coffee distributor makes methods in the *CoffeeOrderIF* interface available to prospective customers

```
package coffees;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface CoffeeOrderIF extends Remote {
  public Coffee[] getPriceList()
        throws RemoteException;
  public String orderCoffee(String coffeeName, int quantity)
        throws RemoteException;
}
```

- There is a *Coffee* object for each coffee that the distributor currently has available for sale
- The method *getPriceList()* returns an array of *Coffee* objects, each of which contains a *name* field and a *price* field
- The method *orderCoffee()* returns a *String* that confirms the orders or states that the particular coffee is on back order

Example: Coffee Order Implementation

```
package coffees;
public class CoffeeOrderImpl implements CoffeeOrderIF {
  // Query the company's database to get the current information
  // and return the result as an array of Coffee objects
  public Coffee[] getPriceList() throws RemoteException {
  // Implementation omitted
  }

  // Query the company's database to see if the particular
  // coffee is available in the quantity specified.
  // If so, set the internal order process in motion and send
  // a response to the customer that the order will be filled
  // If not, place an order to replenish the supply and
  // notify the customer that the coffee is back ordered
  public String orderCoffee(String coffeeName, int quantity)
        throws RemoteException {
  // Implementation omitted
  }
}
```

Packaging a Web Service in a WAR File

- A **Web Application Archive (WAR)** file is a JAR file that contains all of the files needed for the Web application in compressed form
  - For example, the *CoffeeOrder* Web Service might be packaged in the *jaxrpc-coffees**.war*** file
- An **XML Deployment Descriptor** file, named ***web.xml***, that contains information needed to deploy the service, must be present in the WAR file
  - For example, if a Servlet engine such as Tomcat is used, the XML Deployment Descriptor includes the Servlet name and description, the Servlet class, initialization parameters, and other startup information
- A **Configuration** file is one of the files, referenced in a *web.xml* file, that is automatically generated by the mapping tool
  - For example, for the *CoffeeOrder* Web Service, the Configuration file is named *CoffeeOrder**_Config.properties***

Deploying a Web Service

- Deploying the *CoffeeOrder* Web Service within a **Servlet container in Apache Tomcat** is accomplished by copying the *jaxrpc-coffees.war* file to the Tomcat *webapps* directory
- Deploying the *CoffeeOrder* Web Service within an **EJB container in an EJB/J2EE application server** is facilitated by deployment tools supplied by the J2EE application server vendor

Coding the Client

```
package coffees;
public class CoffeeClient {
  public static void main(String[] args) {
    try {
      CoffeeOrderIF coffeeOrder =
          new CoffeeOrderServiceImpl().getCoffeeOrderIF();
      Coffee[] priceList = coffeeOrder.getPriceList();
      for (int i = 0; i < priceList.length; i++) {
          System.out.print(priceList[i].getName() + " ");
          System.out.println(priceList[i].getPrice());
      }
      catch (Exception ex) {
      ex.printStackTrace();
} } } }
```

- The class *CoffeeOrderServiceImpl*, which is generated by the mapping tool, is a **stub factory** that creates an instance *coffeeOrder* of *CoffeeOrderIF*
- The method *getPriceList()*, which is a method of *coffeeOrder*, returns *priceList*, which is then printed

Invoking a Remote Method

- First, the client must discover the Web Service
  - The JAX-RPC runtime can determine the *CoffeeOrder* **Web Service URI** from its **WSDL description**
  - If a WSDL document is not used, you need to supply the **Web Service URI** as a **command-line argument on the client-side**
- Then, you can invoke a method of the Web Service, e.g., to invoke the *getPriceList()* method of the *CoffeeOrder* Web Service, you would type on the command-line

  java coffees.CoffeeClient
  - This allows you to make a **static remote method call at compile time**
  - It is also possible to make a **dynamic remote method call at run time**, using a dynamic proxy or the Dynamic Invocation Interface (DII), although doing so is expensive in performance and resources used