Essential Concepts of Web Services

Dr. Lixin Tao

http://csis.pace.edu/~lixin
Computer Science Department
Pace University

July 12, 2005

Table of Contents

1 Why is enterprise system integration important? 2
2 What are the key challenges to enterprise software systems integration? 2
3 What is Service-Oriented Architecture (SOA)? 3
4 What is Web Service? 3
5 Why do we need Web services? 4
6 What kind of applications can Web services support? 5
7 What are the alternative technologies? 5
8 What is the four-tiered Web architecture? 6
9 What are HTML forms? 7
10 What are the GET and PUT requests of the HTTP protocol? 9
11 What are the basic differences between HTTP GET and HTTP POST? 10
12 What are the basic functions of a Web server ASP/servlet/JSP component? 11
13 What is the basic design of a Web service? 12
14 What are the major components of the Web service technology? 14
15 Why is XML playing an important role in the Web service technology? 15
16 Is Web service a proprietary technology? 16
17 Is Web service a mature technology? 16
18 How to secure Web services? 16
19 How to boost the performance of Web services? 16
20 Why should we learn Web services for both Java and .NET? 17
21 Course coverage for *Enterprise System Integration with Web Services* 17
22 How can I jump-start to learn hands-on Web services now? 18

[2] This document is available at http://csis.pace.edu/~lixin/pclc/ws/webServiceConcepts.pdf. It has an accompanying lab manual at http://csis.pace.edu/~lixin/pclc/ws/webServiceWorkshopLab.pdf. This document is the main course material for the Pace University *Enterprise System Integration with Web Services* workshop.

1 Why is enterprise system integration important?

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.

2 What are the key challenges to enterprise software systems integration?

This course aims at addressing the major challenges in the integration of today's enterprise software systems. *Enterprise software systems* refer to large software systems deployed at companies whose reliable and efficient operation is critical to the business of the companies.

The major characteristics of today's enterprise software systems include:

- Many of them are hosted on application servers like Java's Enterprise JavaBeans (EJB) servers (typical examples are *WebLogic* from BEA and *WebSphere* from IBM) and Microsoft's *Transaction server*; and they support remote clients, either within the same company or from other companies.
- Today's e-commerce asks for not just efficient business to client (B2C) transactions, but also efficient business to business (B2B) transactions. Example B2B transactions include supplier/consumer transactions between two companies: one company produces one commodity, and another company needs to buy this commodity to support the production of its own commodities. While B2C transactions are usually conducted through Web browser interfaces, B2B transactions are usually conducted between the software systems belonging to different companies; no human beings are directly involved. Enterprise system integration is the trend of the IT industries.
- No hardware platforms (mainframe computers, high-performance workstations, PCs, Macintoshes, etc.) can completely dominate the enterprise computing market. No software platforms (Windows, Unix, Linux, Java virtual machines, etc.) can completely dominate the enterprise computing market. In B2B enterprise computing, the interacting software systems, belonging to two different companies, may be implemented on

different hardware/software platforms and in different languages. Many existing mature networking technologies, like Java's *Remote Method Invocation (RMI)*, Microsoft's *COM+* or *.NET Remote*, can only support networking among communicating parties running on the same platform.

- Enterprise software systems are usually protected from external intruders by network firewalls. But by the default, the firewalls only allow messages passing through port number 80 for external clients to access its internal Web servers, which is intended to be public for making commercials, and other port numbers specially configured by the firewall administrators. Existing mature networking technologies, like Java's Remote Method Invocation (RMI), Microsoft's COM+ or .NET Remote, or OMG's CORBA (for more information, visit http://www.omg.org), all need to open TCP/IP communication sockets on the servers running on special port numbers not allowed by firewalls by default.

Therefore we are facing the fundamental problem: How can we let enterprise software systems, implemented and running on heterogeneous platforms, interact with each other and invoke each other's methods over the Internet? Or put it another way, how to make these heterogeneous systems interoperable?

3 What is 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.

4 What is Web Service?

Web service is a special term referring to a particular technology of making heterogeneous software systems interoperable. It is currently the main implementation technology for *Service-Oriented Architecture*. Suppose a system is implemented in COM+ and running on Windows XP. How can a Java program remotely invoke its public methods?

Web service's main objective is to support remote system integration. But this support is mainly at the primitive remote method call level. Performance is not the main objective. Web service is based on technologies standardized by industry consortiums like W3C (http://www.w3.org/). It is not a proprietary technology. It is fundamentally a *wrapper technology* in the sense that we normally don't create a new software system from scratch based on Web services. We normally already have business logics implemented in suitable technologies like EJB or COM+, and we wrap up the existing implementation through a Web server and a few new Web server components (automatically generated by tools) so the selected public methods of the existing implementation can be remotely invoked by client systems running on any platform. Client systems accessing Web services don't need to install special client software; they only need to have some XML processing abilities, which are publicly available and will be, if not already, integrated into operating systems or programming language libraries.

Since the main objective of Web services is for providing remote access to existing server methods, Web services cannot improve the performance of the existing systems. Our best hope is to minimize the performance overhead introduced by the extra wrapping layer between client systems and the server business logics.

Microsoft is one of the key proponents of Web services. Microsoft is, up to now, dominating the PC world. It is extending its territory to enterprise computing. For many years Microsoft tried to port its mature COM/DCOM technologies to Unix/Linux so that applications running on Unix and Windows platforms can work together. But it turns out the Microsoft Transaction Server, the application server supporting COM/DCOM/COM+, is inherently depending on the PC platform, and Microsoft finally gave up the porting plan. Instead Microsoft started to consider how to support *interoperability*, a more humble objective, between applications designed for and running on Windows and Unix platforms. It first proposed the idea of Web services, and the idea was soon embraced by all the major IT companies including IBM and Sun. In the new Microsoft .NET technology, Web services are becoming its flagship component. Today Web services have been supported by all platforms.

As a wrapping technology, Web services are not going to replace the major existing server technologies like EJB, COM+, or CORBA. It only extends them with interoperability among heterogeneous platforms. Whether a Web service performs well heavily depends on which server technology is used to implement its business logics.

5 Why do we need Web services?

The question is already answered in "What is Web Service". Basically, the existing server technologies failed to support efficient interoperation among heterogeneous platforms, or they failed to dominate the IT industries (CORBA is an example, which can support objects implemented in different languages and running on various platforms to invoke each other's methods. Sun's Java RMI and EJB are based on CORBA. But CORBA failed to be accepted by the majority of the IT industries due to many business and technological reasons. CORBA is now

mainly used by big IT companies to support mission critical tasks). Web services is a nonproprietary wrapper technology that enables (existing) objects running on heterogeneous platforms to invoke each other's methods.

6 What kind of applications can Web services support?

Web services are mainly for business to business (B2B) applications. The clients of a Web service are usually programs, not human beings.

If the servers and clients of an application will all run on the same platform and be implemented in the same programming language, then there is no reason to use Web services. As we explained earlier, Web service may introduce performance penalties. For example, if both the servers and clients of an application will only use Windows platform, then they should definitely adopt Microsoft .NET technologies. If all the clients and servers of an application will be implemented in Java but may be on different platforms, then we should definitely adopt J2EE technologies.

If the business logics of a server application needs to be accessed very frequently (like one method call every 10 milliseconds) by clients implemented in different languages and running on various platforms, and the scalability of such a server application (the *scalability* of a server application is how many concurrent clients the server can support with good performance; for example, how many clients can simultaneously access a Web server while not noticing obvious delays) is critical, then Web services may not be a suitable technology to use due to the performance penalties they suffer. CORBA may be a better candidate for this scenario.

If the business logics of a server application needs to be accessed by heterogeneous clients at a low frequency, and it is desirable to minimize the burden for client software installation and server firewall configuration, then Web services may be a good choice to be adopted.

The following is a simple application example of Web services. An online bookstore may use Web services to expose some methods for client software to enquire about its book supplies and conduct book sale transactions. A book broker like http://www.bookfinder4u.com may implement a book brokerage service for its human clients. When the clients ask book broker bookfinder4u for the best price for a book, bookfinder4u will search for prices for this book at various online bookstores as their clients through their Web service methods, and then report back to its own human client which online bookstore is providing the best price for the book.

7 What are the alternative technologies?

For Java platform, remote method invocation (RMI) allows Java objects to call remote methods running on a remote Java virtual machine. But RMI only works on Java platform. RMI is implemented on top of CORBA technology.

For Windows platform, COM+ and .NET Remote can support remote method invocations between COM+ objects running on different Windows server machines.

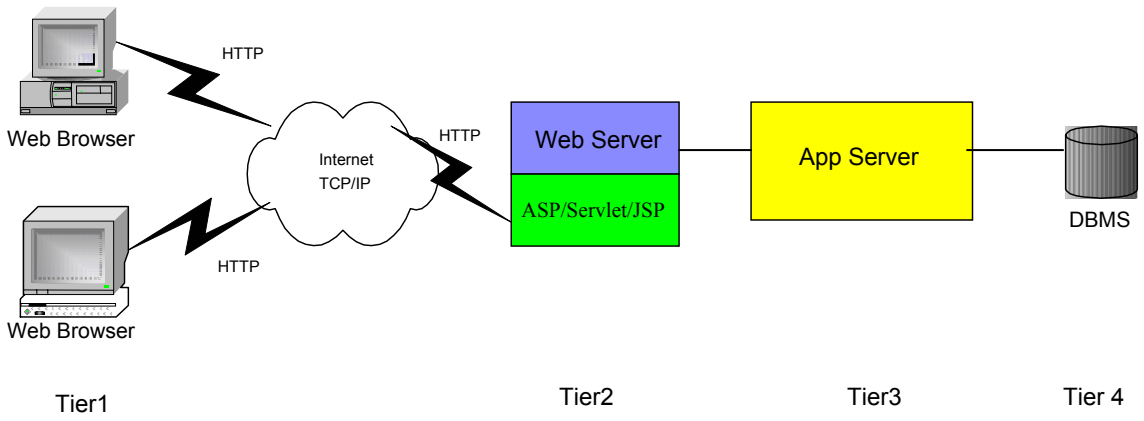
Common Object Request Broker Architecture (CORBA) is a mature technology supported by over 200 IT companies of the Object Management Group (www.omg.org). Its development started in 1990, and it aims directly at efficient enterprise software system integration with

distributed objects. CORBA can support objects implemented in different languages and running on different platforms to invoke each other's methods. CORBA also supports method call backs (a client passing a reference of a method or object through one of the parameters of a remote method call to a server, and the server later can use that client method or object to notify the client of the advent of some server event), which is missing from Web services so far. But due to business and technological reasons, CORBA has not been adopted as the mainstream technology for application-level networking. CORBA is also a complex and expensive technology.

8 What is the four-tiered Web architecture?

Even though this course is not for teaching Web applications like *servlet*, *JavaServer Pages* (*JSP*), or *ASP .NET*, which are human-oriented or browser-oriented Web technologies, the implementation of Web services depends on a Web server supporting one of these technologies. Web architecture is the carrier for Web services, as you will see clearer after I answer the question "What is the basic design of a Web service?"

Today's Web applications are based on a four-tiered client/server architecture, as shown below:



At *tier 1* on the client side, a user uses a Web browser to interact with a remote Web application running on the Web server. The Web browser can communicate with the Web server through the HTTP protocol running on top of the Internet. As I will explain later, the Web browser sends HTTP GET or POST requests to a Web server for sending client information to the Web server and requesting information to be sent back, and the Web server will then return an HTTP response, which normally contains an HTML file. An HTML file contains the data the client requested, as well as instructions on how to present the data visually to the client. A Web browser understands how to interpret an HTML file to provide the user with a graphic user interface (GUI). Therefore, tier 1 is also called the *client-side presentation tier*.

Web server is mainly responsible for returning HTML files to Web browsers. The original Web server can only serve static HTML files stored on a hard disk. Due to the demand for customized services, Web servers are now extended with servlet containers (like Tomcat) or ASP containers (IIS Web server) to support ASP/servlet/JSP technologies to generate HTML files on the fly, based on user information and data in databases. The Web server is considered as *tier 2* of the Web architecture. Tier 2 is also called the *server-side presentation tier*, since it mainly generates HTML files for the client-side browser to render.

Servlet and JSP are Java technologies for extending a Web server. ASP is Microsoft's technology for extending a Web server. Servlets are Java classes. JSPs are based on servlets, but they are HTML files with some embedded Java code, so they are easier for Web developers to use. ASP is basically Microsoft's version of JSP. ASP, servlet, and JSP are replacing older technologies like CGI because they use threads, instead of processes, to process requests from the remote clients, thus leading to better performance.

Tier 3 is responsible for business logic computation. For a simpler Web application, business logic computation can be done in tier 2 by the ASP/servlet/JSP components. But for heavy-duty Web applications, a separate application server is normally used to run the business logics. Enterprise JavaBeans (EJB) and COM+ are major examples for application server technologies. The application servers will expose some methods for the Web server to call. When an ASP/servlet/JSP component needs computation for generating an HTML file, it will invoke a proper application server method. If the business logic needs data, it will access tier 4, the database tier. The application servers are usually also responsible for in-memory data caching for data in the databases to improve the efficiency of database accesses.

Tier 4 is normally the databases, but it can also be legacy systems of a company that were developed on an older technology but are still providing important services to the company.

The separation of tier 2 and tier 3 of a Web application is mainly for improving the Web application's *scalability*: how many concurrent clients can get fast services. The Web server and the application server can run on different computers to take advantage of more CPUs. A Web server or application server can also be duplicated on multiple hardware server machines to further improve the Web application's scalability. Here we try to balance the work load on a cluster of computers. For example, a Yahoo data center typically has around 50 clustered computers all running Web servers, and a client's HTTP request can be served by any of these Web servers transparently.

9 What are HTML forms?

A Web browser uses HTML forms to collect information from a user, and sends the information to a Web server for processing.

The following is an example HTML form:

```
<html>
<body>
    <form method="post" action="http://csis.pace.edu/survey/input">
        Enter your name: <input type="text"  name="user">
        <input type="submit"  value="Submit">
    </form>
</body>
</html>
```

When you load this file into a Web browser, you will see a GUI like the following:

HTML element (tag) "form" can contain text and nested elements. An "input" element with attribute "type="text"" declares a textbox, and its "name" attribute defines a variable name that the Web server components can use to retrieve the value users enter into this textbox. For example, "<input type="text" name= "user">" declares a textbox whose value (user input in this textbox) will be retrieved on the Web server through a variable named "user". An "input" element with attribute "type="submit"" declares a submit button, and its "value" attribute specifies the string to be displayed on the button. When the user clicks on the submit button, all information the user entered in the form will be submitted to a Web server component specified by the "action" attribute of the form. For example, "<input type="submit" value="Submit">" will present a submit button with word "Submit" displayed on the button surface. When a user clicks on this button, the string that the user typed in the textbox will be submitted to "http://csis.pace.edu/survey/input", which is Web component "input" of Web application "survey" running on the Web server computer with a domain name "http://csis.pace.edu".

HTML element (tag) "form" supports attributes "method" and "action." Attribute "method" normally takes constant values of either "get" or "post", and they specify whether the form information will be sent to the Web server through HTTP GET or POST requests, which are explained below. Attribute "action" specifies the Web server component that will receive the form information and process it. After the Web server component processes the form information, it will return an HTTP response to the Web browser, which may contain the status of the form submission and an HTML file.

The value of attribute "action" is typically a *Uniform Resource Locator (URL)*. It identifies (1) the domain name of the server computer, say "http://csis.pace.edu"; (2) the port number that the Web server is listening to on this server computer, say "80", which is separated from the domain name by a colon (:), leading to "http://csis.pace.edu:80" (this shows the Web server is running on port number 80, which is the default Web server port number and can be omitted; other port numbers can be used too; a server computer can run multiple networking server applications at the same time as long as each of them uses different port numbers to make itself unique); (3) the Web application running on this Web server, say "survey" (normally corresponding to the name of the file system directory holding this Web application, under "C:\Inetpub\wwwroot" for our IIS Web server installation, or under "TomcatHomeDir\webapps" for Java Tomcat Web server), leading to "http://csis.pace.edu:80/survey"; and (4) the name of the Web server component that will process the form information, say "input", which is usually an ASP .NET page for Microsoft IIS, or a servlet or JavaServer Page for Java Web servers like Tomcat, leading to a URL of "http://csis.pace.edu:80/survey/input". A Web application normally contains many Web server components.

10 What are the GET and PUT requests of the HTTP protocol?

When a user clicks on a submit button of an HTML form, or clicks on a hyperlink in an HTML file, the Web browser will generate an HTTP request to a Web server, and the Web server will return an HTTP response.

HTTP is an application-level protocol, running on top of the TCP/IP transportation protocol. While TCP/IP supports the illusion of a reliable virtual communication channel between any pair of computers on the Internet, HTTP specifies how Web browsers and Web servers are supposed to exchange information through the TCP/IP virtual communication channels. HTTP protocol is very simple. It is always the Web browsers that initiate communications; Web servers can only respond to Web browsers' requests. When a Web server receives an HTTP request, it will return an HTTP response, and then forget about the request.

There are two types of HTTP requests that are important for Web applications: GET and POST. HTTP GET was originally designed for downloading HTML files from a Web server, and HTTP POST was originally designed for submitting user information (mainly form information) to a Web server. But both GET and POST can be used for both client information submission and server file downloading. Their differences will be summarized later in this section.

When a user clicks on an HTML form with value "post" for its attribute "method", an HTTP POST request will be issued to a Web server component. The following is an example HTTP POST request:

```
POST   /survey/input   HTTP/1.0
Accept:  text/html
Accept:  audio/x
User-agent: MacWeb

user=Ada
```

On line 1, "POST" indicates that this is an HTTP POST request; "/survey/input" indicates that the Web browser needs to submit data to the Web component, with alias "input", of the Web application "survey" on the Web server for processing; and "HTTP/1.0" tells the Web server that the Web browser supports HTTP version 1.0, which is now still the dominant version in use.

Lines 2, 3, and 4 are examples of "header lines", each being a name/value pair separated by a colon (:). Line 2 declares that the Web browser can present HTML files, which is a special kind of text file. Line 3 declares that the Web browser can present any type of audio files. Line 3 declares that the Web browser is based on a particular Web browser architecture called "MacWeb". After a blank line is the *entity body* for the HTTP request, which holds the data to be submitted to the Web server. The entity body of an HTTP request can hold data of any type and of any length.

Upon receiving such an HTTP POST request, the Web server may return an HTTP response similar to the following:

```
HTTP/1.0   200   OK
Server: NCSA/1.3
Mime_version: 1.0
Content_type: text/html
Content_length: 2000

<HTML>
……
</HTML>
```

On line 1, "HTTP/1.0" tells the Web browser that the Web server supports HTTP version 1.0; "200" is the status code indicating that the HTTP request was OK or successful; "OK" is a brief explanation of status code "200".

Lines 2, 3, 4, and 5 are header lines. Line 2 indicates the architectural category of the Web server. Line 3 indicates the capability of the Web server to handle various media types (MIME is for "Multipurpose Internet Mail Extension", an industry standard originally designed to extend mail server's ability to process multimedia email attachments). Line 4 indicates the data type for the data after the header lines. For this example, the HTTP response contains an HTML file. Line 5 specifies, starting from the first byte after the blank line below the header lines, in this case character "<", how many bytes really belong to the HTTP response data. In this example, the attached HTML file contains exactly 2000 bytes, from the starting "<" to the ending ">". This contents length makes the HTTP data communication more "noise"-resistant.

There will always be a blank line immediately after the header lines for indicating the start of the content data, called *entity body*.

11 What are the basic differences between HTTP GET and HTTP POST?

When a user clicks on a hyperlink or the submit button of an HTML form with attribute "method="get"", HTTP GET will be used to submit data to a Web server. HTTP GET uses *query string* to pass user information to the Web server, as in the following example:

http://csis.pace.edu/survey/input?user=Peter&age=22

In this URL, "survey" is the name of a Web application, "input" is the name of a Web server component for processing the user data, and the substring starting from "?" is called a *query string*, which is a sequence of name/value pairs separated by "&". In this particular example, the user informs the Web server that variable "user" has value "Peter", and variable "age" has value "22". This URL may be hard-coded in a hyperlink, or it can be generated by a form like the following:

```
<form   method="get"  action="http://csisi.pace.edu/survey/input">
     Name:  <input  type="text"  name="user">
     Age:    <input  type="text"  name="age">
     <input  type="submit"    value="Submit">
</form>
```

When the user clicks on the submit button of a form using method "get", all name/value pairs, in this case variable names "user" and "age" and their actual values the user typed in their corresponding textboxes, will be concatenated to the URL specified by the "action" attribute of the form, after a question mark "?", and separated with each other by character "&".

On the other hand, if an HTML form uses method "post", such user data will be passed to the Web server through the *entry body* of the HTTP request, which is the data after the blank line immediately following the header lines.

The user data passed to a Web server through a query string is first stored on the Web server's data buffer named QUERY_STRING (an operating system environment variable defined in *Common Gateway Interface*, or *CGI*). Since this buffer has a limited capacity of around 512 bytes, passing user data larger than 512 bytes through query string may lead to user data truncation, or even worse, server crash. Therefore, if an HTML form needs to send large amount of data to a Web server, HTTP POST should be used, since with HTTP POST data will be passed through the request entity body and read by Web server components incrementally through an input stream, which enforces no limit on data size.

If the user data of an HTML form is sent to a Web server through HTTP GET, all user data will be appended to the target URL as a query string, thus people may read it over the user's shoulder; the user may copy a big chunk of data into a textbox thus submitting over-sized data and crashing the server. Files downloaded by HTTP GET will by default be cached on the user's PC for more efficient future access to the same file. If some server data changes dynamically, it is better off to use HTTP POST to download the data to avoid such data caching.

If a user needs to submit large amount of data to a Web server, or the user doesn't like the user data to appear as a query string on the URL, or the user needs to download some dynamic information from a Web server, then HTTP POST should be used to access the Web server.

12 What are the basic functions of a Web server ASP/servlet/JSP component?

In this document I use the term *Web server component* to denote either *Active Server Pages (ASP) .NET* from Microsoft, or *servlets* and *JavaServer Pages (JSP)* for the Java platform. They are all mainly designed to process HTTP GET/POST requests.

When a user clicks on a hyperlink in an HTML document or clicks on the submit button of an HTML form, the Web browser generates an HTTP GET/POST request, as described in the previous two sections. The Web server will first decompose user data, including the request's entity body, the header lines and cookies (cookies are name/value pairs saved by the Web server

on the client computer earlier), the Web browser's HTTP version, the query string appended to the URL, the request method (GET or POST), the content type, and store them in proper operating system environment variables called *Common Gateway Interface (CGI)* variables. The Web server may also process the client data and present it in object-oriented way to its Web server components. A Web server component will read the data of the HTTP request, either retrieve a static file from its file system or dynamically generate a file (normally HTML file) based on data from the client and data in a database, and send the file back to the Web browser as part of its HTTP response. HTTP response can return any type of files as its entity body.

HTTP requests can be generated by Web browsers; they can also be generated by programs. Languages like Java and C# both have *Application Programming Interfaces* (APIs), or methods, to support the issuing of HTTP requests to Web servers and the processing of HTTP responses. Web services take advantage of this capability of the programming languages.

13 What is the basic design of a Web service?

Web service is a misleading term. Its objective is very simple, that is to enable programs to invoke methods of remote objects implemented in various programming languages and running on various operating system and hardware platforms. Web service only resolves the *interoperability* of heterogeneous computing systems, which is a low-level basic service for enterprise system integration. Other important technologies for enterprise system integration include XML for data integration. Web service security is mainly provided by separate technologies working at transportation protocol (like SSL) or application levels (like SOAP data encryption). Even the humble interoperability of Web services only provides the simplified connectivity service of mature technologies like CORBA for enterprise system integration. The functions supported by Web services are common denominators of the major distributed computing technologies, not the improvement of them. The strengths of Web services include its non-proprietary nature, no client software installation for each Web service, and easy configuration in firewall environments.

When a service provider decides to expose some public methods of its local objects (class objects, COM+ objects, or EJB objects), it needs to identify the signatures of these methods, and sometimes list these signatures in an interface file. The signature of a method includes its name, number of parameters, type of each parameter, method return type, and exceptions that may be generated by the method body. In .NET, we just need to add method attribute "[WebMethod]" in front of each of these methods. In Java, we write a Java interface file containing these methods' signatures.

A utility (like "wsdl.exe" or "wscompile.exe") will take the method signatures as input, and generate a Web server component as the entry point of client method invocations, and a *Web Service Description Language (WSDL)* file, which is an XML file describing the signatures of the exposed methods and the data types used in the method signatures. The Web server component, either an ASP or servlet, will be deployed on a Web server of the service provider, ready to receive HTTP POST/GET requests representing client method invocations. The WSDL file will be distributed to potential clients in various ways. A typical approach is to register the WSDL file with a registry service (yellow pages for Web services) so that the clients can use keywords to search for these Web services and download the WSDL files of the chosen Web

services. The WSDL file is the only file a client needs to invoke methods described in the WSDL file.

When a client needs to accomplish some tasks with Web services, he/she will typically search the Web service registries with keywords to find descriptions of various Web services that may resolve the tasks, compare their prices and performance if such information is available, and then download the WSDL file for one of the Web service providers. Another utility will take this WSDL file as one input, and the name of the programming language for the client code as the another input, and generate the source files for a proxy class for the Web services described in the WSDL file, as well as all the necessary data types needed by method calls to those Web service methods, in the programming language specified as input to the utility. For example, suppose the Web services are implemented in C# on IIS, and the client system is in Java. Then the proxy class and its supporting classes will be generated in Java. The proxy class is mechanically generated by a utility. It exposes exactly the same public methods as those described in the WSDL file, which are the same as those exposed on the Web service provider's server machines. The client can now compile the proxy class and generate an instance of the proxy class for invoking the remote Web service methods.

When the client system invokes a method of the proxy class, the body of the method will generate a *Simple Object Access Protocol (SOAP)* message, where SOAP is a dialect of XML for describing all information for a method call (name of the method, argument values, return value, etc.); generate an HTTP POST/GET request to the entry point of the Web services on the target Web server, which is defined in the WSDL file; and attaching the SOAP message to the HTTP request as its entity body. A SOAP message is just an XML representation of a method call. Its value lies in SOAP's independence from programming languages.

When the Web server component, or the Web service entry point, receives the HTTP POST/GET request, it will retrieve its attached SOAP message, restore the method name and argument values in the language supported by the Web service implementation, and invoke locally the Web service implementation object's actual method with the same signature. Upon receiving the return value from the local method call, the Web server component will rewrite the method return value into another SOAP message representing the return value in a language-independent way, and pass the new SOAP message as the entity body of its HTTP response to the method body of the client-side proxy method, which will unwrap the SOAP message and recover the method return value in its own language, and return this value back to its local invoker as the proxy method's own return value. As a result, the invoker of the proxy method does not need to have knowledge of what is happening behind the scene; to the Web service invoker, the proxy object seems providing services locally.

Web service is based on the *Proxy Pattern*, which is popular in *Remote Procedure Call (RPC)* or *Remote Method Invocation (RMI)* implementations. The HTTP protocol is used by the Web services as a bridge between its client systems and service providers. A Web service provider must run a Web server to receive HTTP POST/GET requests for Web service invocations. The client-side system only needs the capability of generating proxy and its supporting classes in its local language, and processing XML files in WSDL and SOAP.

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 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.

14 What are the major components of the Web service technology?

Based on our description above, we can see that 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.

  Microsoft has its proprietary Web service registry mechanism called DISCO. While it is convenient to use on Windows platform, it is not supported by the broader industry practices.

  At this point there are no commercial UDDI registries available. Some companies are running their public free UDDI registries to train the future clients and try out the technology. http://uddi.microsoft.com and http://uddi.ibm.com are two of such test UDDI registry implementations. UDDI registry is not critical for learning Web services, since it is only one standardized way for clients to get the service provider's WSDL files.

- *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 entry 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. Because XML is not needed explicitly in the development of Web services and there are separate courses for XML, this course will only make basic introduction to XML.

- *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, even

though this course is not supposed to teach Web technologies like ASP .NET, servlets, or JavaServer Pages.

15 Why is XML playing an important role in the Web service technology?

XML is an extensible markup language for describing data structures. While HTML is for presenting data graphically, XML focuses on logical structures among the data elements. The type of elements allowed in an XML file can be added as justified, as the name "extensible" implies.

An XML document is made up of a set of elements. Each element starts with a *start tag*, like <html> and ends with an *end tag*, like </html>. Between the start tag and the end tag there may be data or nested elements, as in HTML. The start tag of an element may carry some attributes, like <a href="http://csis.pace.edu/~lixin" >, where "href" is an attribute of tag "a", its value is "http://csis.pace.edu/~lixin". An XML document can contain exactly one top-level element. For example, in an XHTML file, which is a special type of XML files equivalent to HTML, the top-level element is always marked up with tags <html> and </html>.

The following is a simple example of an XML file.

```
<?xml version="1.0"  encoding= "utf-8"?>

<message  from="Joe"  to= "Sue">
   <date>September 9, 2004</date>
   <body>
      Whatever message body, which may contain nested
      elements like <b> bold face</b>.
   </body>
   <attachment>
      Whatever attachments to the message.
   </attachment>
</message>
```

In this example, the first line is called an *XML prologue*, which declares the XML version adopted by this document and the encoding of its data. While at this time we only have version 1.0 for XML specification, having such a declaration allows the future consumer applications of this document to know its limitations.

This document has <message> as the start tag for its top-level element. Tag <message> supports two attributes, namely "from" and "to", specifying the message's sender and receiver. This particular message contains three nested elements starting with tags <date>, <body> and <attachment>, and the <body> element in turn also contains a nested element marked up with tag <b> for bold face.

Since XML is extensible, application-independent, and can be easily parsed with publicly available parsers, XML is suitable for describing the structures of particular types of documents. For example, we can define a type of XML dialect that can be used to describe the signatures of

methods to be exposed by Web services. For this purpose, we need to standardize what types of elements are allowed in such a document, how the elements can be nested, in which order they must appear, which attributes that each element can support, what type of data can go between the start tag and the end tag of an element, etc. Such syntax definition of a type of XML documents is usually done through a *Data Type Definition (DTD)* language, or more recently, through *XML Schema*, which in turn is also an XML document by itself. Since the structure of a WSDL file is standardized, WSDL files can be generated and consumed by WSDL utilities from different companies. Here we can see that XML itself has limited applications. The full power of XML will be realized when the document types are standardized by involved users, or by government or industry consortiums. But it is always very hard for the industries to agree on a particular standard, partially due to technical reasons, and mainly due to market competitions.

16 Is Web service a proprietary technology?

The answer is no. Web service is based on specification standards set up by industry consortiums, in particular W3C. Its non-proprietary nature gives Web services the widest potential application domains and the least resistance to its adoptions.

17 Is Web service a mature technology?

The answer is no. At this time Web service is still very young and immature. For example, it does not support method callbacks, and the support for parameter passing for complex data types is not complete.

18 How to secure Web services?

While many Web services are open to 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 services and their client systems. As a result, intruders cannot easily eavesdrop data between the Web services and the client systems.
2. Encrypt SOAP data by the client-side's proxy classes, and decrypt SOAP data at the Web server component end. 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.

19 How to boost the performance of Web services?

Web services by itself only resolves the problem of interoperability of heterogeneous information systems. The performance of a Web service depends mainly on the implementation technology of the business logics and marginally on the performance of the Web server that acts as the entry-point of requests for Web services.

For focusing on the fundamental concepts of Web services, a course or book on Web services normally use a simple object to implement business logics. While this is perfectly fine and simple, such business logic implementation is not taking advantage of modern application server technologies for scalability and other advanced features. If the performance is important for a Web service, the business logic behind it should be implemented on a mature application server technology like Enterprise JavaBean or COM+. Therefore, Web services are complementing the existing application server technologies of J2EE and .NET, instead of replacing them. If you are a serious software developer, you need to learn both Web services and J2EE or .NET application server technologies.

20 Why should we learn Web services for both Java and .NET?

First, Visual Studio .Net provides very high-level abstraction of the Web service implementations, so the students can focus more on fundamental ideas instead of on language API details.

Second, Microsoft .NET represents a very competitive computing platform rich in job opportunities. C# is very similar to Java, so students don't need to make significant effort in learning a new language. Understanding both Java and C# will enable students to put their knowledge of Java in perspective.

Third, the main objective of Web services is for bridging heterogeneous platforms, and it is much more interesting if students can see how programs written in different languages or running on different platforms can interact with each other.

21 Course coverage for *Enterprise System Integration with Web Services*

1. Complete coverage of fundamental concepts of the Web services technology and its supporting technologies including Web architecture, HTTP, servlet and ASP, and XML.
2. Web service architecture, SOAP, WSDL, UDDI, and SOA.
3. Major development tools for Web services, including *J2EE 1.4 SDK*, *WebSphere*, *WebLogic*, and *Microsoft Visual Studio .NET* .
4. Synchronous and asynchronous Web services.
5. Design and implementation of Web services and their client systems in Java.
6. Introduction of *C# .NET* for Java programmers.
7. Design and implementation of Web services with *Microsoft Visual Studio .NET*.
8. Design and implementation of application and Web clients for Web services with *Microsoft Visual Studio .NET*.
9. Interoperation of Web services across Java and .NET platforms.
10. Fast transformation of existing applications for supporting Web services.
11. Web service security.
12. Web service performance and scalability.
13. Introduction to boosting performance of Web services with *Enterprise JavaBeans*.

The prerequisites for taking this course are basic programming knowledge in Java, and a strong desire for hands-on learning of the latest IT technologies.

22 How can I jump-start to learn hands-on Web services now?

This tutorial has an accompanying well-designed lab session for you to explore hands-on development of Web services and their consumer applications on both Java and Microsoft .NET platforms. You will develop Web services on both Java and .NET platforms, develop standalone and Web clients for your Web services, and let Java clients invoke .NET web services that run either locally or remotely across the Internet. The complete lab manual is at

http://csis.pace.edu/~lixin/pclc/ws/webServiceWorkshopLab.pdf.

The installation guide for Java tools for Web service development is at

http://csis.pace.edu/~lixin/pclc/ws/JavaWebServicesToolInstallation.pdf.

The installation guide for .NET tools for Web service development is at

http://csis.pace.edu/~lixin/pclc/ws/DotNetWebServicesToolInstallation.pdf.