Lecture 11

Web Services
HTTP, SOAP, WSDL, UDDI
The Problems

• Provide a mechanism for integrating together systems built on:
  – Different languages
    • C, C++, VB, Delphi, ...
  – On different hardware & OSes
    • Win, Linux, Solaris, Mac, ...
  – Different LANs even between businesses behind firewalls
    • What exactly is a firewall? (in a minute)
The Problems (2)

• That can be found easily & programmatically
  – Like yahoo! for software applications
• That is independent of the component model used by the applications communicating
  – COM, COM+, CORBA, J2EE/EJB, …
SOA

• Service Oriented Architecture
• High level “services” that are exposed to many applications in the enterprise
• Service are NOT object oriented (hence the name – service oriented)
  – Not RMI or DCOM
  – Exchange documents/data not objects
  – High level / large granularity services
SOA (2)

• Examples services:
  – GetCreditScore(Client)
  – CalculateTax(Amount)
  – GetCatalog()
  – TransferFunds(FundsXferRequestDoc)

• Generally a single parameter (msg or document) is passed in and a single parameter is returned.
SOA (3)

• Originally came to prominence in the field of EAI (Enterprise Application Integration)
• Generally implemented using asynchronous messaging technologies like IBM MQ Series (RPC/RMI could be used too).
SOA (3) – asynchronous messaging based
Firewall

• A firewall regulates access to internal systems, generally HTTP servers

• Types include:
  – Protocol filtering
  – Address filtering
  – Proxy based
Firewall Types

• Protocol filtering
  – Allow only: HTTP, SMTP, FTP, TCP, UDP and/or IP
  – Allow only outside in, inside out or allow both

• Address
  – Allow access, on a per protocol basis, to only certain addresses or addresses and ports
Firewall Types

• Proxy
  – An application (proxy) runs between the clients and the servers that can interrogate every packet that comes in or out and limit the types of packets and content of packets that flow through the firewall.
  – It “fakes” the clients into thinking they are directly talking to the server application. The client requests are “proxy-ed” through this application.
Firewalls in Corporations

- Most do NOT allow TCP, UDP, or IP through!
- They allow HTTP on Port 80.
- Often only allowing HTTP clients to be outside and HTTP servers to be inside the firewall.
- How can you have a complex application at one business access services hosted in a different corporate environment HTTP?
  - EJBs?
  - CORBA?
  - COM+?
  - RMI?
  - TCP?
  - IP?
HTTP to the rescue

• If you package up your request in a HTTP request and pass it to an HTTP server that knows you don’t just want a HTML page returned.

• This is the basic idea of a WebService

• It's like the Browser is for humans, it is for application. It allows applications to make GET and POST requests over HTTP to invoke services remotely.
How?

• Like RMI uses serialization to send data, WS uses a marshalling of data from a client data structure/object into an XML format that can be sent over the wire and through an HTTP server.

• This is called a SOAP document

• SOAP is an XML document format for marshalling & unmarshalling

• SOAP
  – Simple Object Access Protocol
  – Service Oriented Access Protocol

(depending on who you talk to - ignore it for the most part)
WebServices (Request Response)
<soap:Envelope
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns:xsd='http://www.w3.org/2001/XMLSchema'
    xmlns:soap='http://schemas.xmlsoap.org/soap/envelope/'
    xmlns:soapenc='http://schemas.xmlsoap.org/soap/encoding/'
    soap:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>

    <soap:Body>
        <n:getRate xmlns:n='urn:xmethods-CurrencyExchange'>
            <country1 xsi:type='xsd:string'>usa</country1>
            <country2 xsi:type='xsd:string'>japan</country2>
        </n:getRate>
    </soap:Body>

</soap:Envelope>
SOAP is:

- Platform, dev language and component model independent
  - you can create it in java EJB and send it to a WS that is written in COBOL if you wanted.
- The serialized form of an object
- Interesting but you as a dev never need directly manipulate it.
- You have tools to get your request/response to/from java and SOAP.
- Don’t worry about the details of SOAP, leave that to the WS “plumbers”
Stubs & Skeletons for WS

• Called a WSDL file
  – WS Description Language
• Tells clients of a service what the interface to the service “looks like”
• Its another type of XML file
• Generally created via vendor tool like java2wsdl that creates a WSDL to match invoking a java file.
<?xml version='1.0' encoding='UTF-8'?>
<!--generated by GLUE on Tue May 21 10:39:36 CDT 2002-->

<wsdl:message name='getRate0In'>
  <wsdl:part name='country1' type='xsd:string'>
    <wsdl:documentation>The country to convert from</wsdl:documentation>
  </wsdl:part>
  <wsdl:part name='country2' type='xsd:string'>
    <wsdl:documentation>The country to convert to</wsdl:documentation>
  </wsdl:part>
</wsdl:message>

(CONTINUED …)
WSDL (CONTINUED…)

... <wsdl:message name='getRate0Out'>
  <wsdl:part name='Result' type='xsd:float'>
    <wsdl:documentation>The exchange rate</wsdl:documentation>
  </wsdl:part>
</wsdl:message>
</wsdl:message>

<wsdl:portType name='examples.publish.Exchange'>
  <wsdl:operation name='getRate' parameterOrder='country1 country2'>
    <wsdl:documentation>Return the exchange rate between two countries</wsdl:documentation>
    <wsdl:input name='getRate0In' message='tns:getRate0In'/>
    <wsdl:output name='getRate0Out' message='tns:getRate0Out'/>
  </wsdl:operation>
</wsdl:portType>

(CONTINUED…)

...
....

```xml
<wsdl:binding name='examples.publish.Exchange' type='tns:examples.publish.Exchange'>
    <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http'/>
    <wsdl:operation name='getRate'>
        <soap:operation soapAction='getRate' style='rpc'/>
        <wsdl:input name='getRate0In'>
            <soap:body use='encoded'
                namespace='http://tempuri.org/examples.publish.Exchange'
                encodingStyle='http://schemas.xmlsoap.org/soap/encoding'/>
        </wsdl:input><wsdl:output name='getRate0Out'>
            <soap:body use='encoded'
                namespace='http://tempuri.org/examples.publish.Exchange'
                encodingStyle='http://schemas.xmlsoap.org/soap/encoding'/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding><wsdl:service name='examples.publish.Exchange'>
    <wsdl:port name='examples.publish.Exchange' binding='tns:examples.publish.Exchange'>
        <soap:address location='http://192.168.1.107:8004/glue/exchange'/>
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

RPC or DOC
WSDL is …

• Platform, dev language and component model independent
  – you can have the same WSDL generated by a WS implemented in java and implemented in COBOL if you wanted.
• Like the Stub & Skeleton created by RMIC
• Like a CORBA IDL or COM typelib/IDL
• You shouldn’t EVER have to manually edit these. Tools should manipulate them for you.
• Ignore the details in most cases. Its just plumbing like RMI is.
How do you “find” a WS?

• Look up its WSDL in a UDDI directory!
• UDDI – Universal Description Discovery & Integration
• Like phone book for WSes.
  – White Pages – by service name
  – Yellow Pages – by service provider
  – Green Pages – by service type
• Like yahoo! Or google for WS Clients.
• Like JNDI for Java or CosNaming for CORBA or the Registry / AD for COMCOM+
• Can be public or private.
UDDI Directory

• A HTTP based process that is published on a well known port and address
• Gets a SOAP request to look up a service(s), returns a response with the information about services (WSDLs)
• Not required. You can just point directly to a WSDL file to get the info needed to invoke a WS.
Creating web services in java. You have many choices:

• The SUN Web Services Developer Pack
  – Big 59 megs+
  – The kitchen sink and more
  – Complex to just run a web service

• GLUE by a company called The Mind Electric (TME)
  – [www.themindelectric](http://www.themindelectric)
  – Simpler, stand alone, quick
  – *You MUST use this for your homework*

• Other tool kits are out there
The glue RUNTIME PLATFORM diagram illustrates various components and services:

- **Plain JAVA**
- **EJB**
- **JNDI**
- **Virtual**
- **Custom**

These components are associated with services:
- **High performance**
- **Small footprint**
- **Easy to use**
- **DOM, SAX, JAXP**

The platform supports XML and includes:
- **Remote deployment**
- **Logging**
- **Auditing & monitoring**
- **Easy configuration**
- **Hot sync**

**JAVA/XML mapping** includes:
- **WSDL**
- **Seamless and automatic, custom mappings without coding**
- **Compatible with .NET, Apache and other platforms**

**SOAP engine** features:
- **Headers, attachments, interceptors, XSLT transformations**
- **Asynchronous/synchronous messaging**

**Transports** include:
- **HTTP**
- **HTTPS**
- **JMS**
- **Custom**

**Servlet Engine** is associated with:
- **GLUE Servlet Engine**
- **Includes web server and JSP engine**

**Any 3rd party application server or servlet engine** is supported.

**Security** includes:
- **JAAS**
- **SSL**
- **HTTP Basic/Digest**
- **Custom**

**UDDI** is used for:
- **Client**
- **High performance server**
- **XML-based storage**

**Dev tools** include:
- **java2wsdl**
- **wsdl2java**
- **java2schema**
- **schema2java**
- **deploy**
- **undeploy**
- **newapp**
- **runapp**
- **enable**
- **disable**
- **enum**
- **parse**
- **holder**

**Console** features:
- **Browser-based**
- **Deployment**
- **Browsing**
- **Invoking**
Glue tutorials
Install Glue

- Download the **standard** edition at [http://www.themindelectric.net/download/](http://www.themindelectric.net/download/)
- Install to your root directory
- Should get a dir structure like:
  - `/electric/glue/` ...
- Look at the files in the `/electric/src/examples/publish` directory first to understand how to get some simple stuff going in GLUE.
Getting glue configured for a command line client

Then perform the following steps:
Add \electric\lib\GLUE-STD.jar (standard) or \electric\lib\GLUE-PRO.jar (professional) to your CLASSPATH as well as the standard Java extension libraries dom.jar, servlet.jar and jnet.jar.
Add \electric\bin to your command path setting. On Windows, this involves updating your PATH environment variable.
To enable SAX, include the library sax.jar.
To enable Jaxen/XPath integration, include the libraries jaxen-core.jar and saxpath.jar.
To enable JSP, include the libraries xerces.jar and jsp.jar, as well as the tools.jar file from your JDK’s /lib directory.
To enable HTTPS for secure web services, include the standard Java extension libraries jsse.jar and jcert.jar.
To enable JAX-RPC, include the libraries jaxrpc-api.jar and saaj-api.jar.
To enable JAXM, include the libraries jaxm-api.jar and saaj-api.jar.

To run the examples without compiling the example source code, add \electric\lib\GLUE-EXAMPLES.jar to your CLASSPATH. Be sure to run all of the examples from their associated source directory.
If you were using a previous version of GLUE, be sure to read the Migration Guide.
Getting NetBeans configured …
Add class path to the IDE

• Your classpath should look something like this:
  
  C:\jdk1.3\lib;C:\netbeans33\lib;..;..;c:\Homework;\electric\lib\GLUE-STD.jar;\electric\lib\dom.jar;\electric\lib\jsp.jar;\electric\lib\sax.jar;\electric\lib\servlet.jar;\electric\lib\xerces.jar;\electric\lib\jnet.jar;\electric\lib\GLUE-EXAMPLES.jar;\electric\lib\jsse.jar;

  C:\jdk1.3\lib;C:\netbeans33\lib;..;..;c:\Homework;\electric\lib\GLUE-STD.jar;\electric\lib\dom.jar;\electric\lib\jsp.jar;\electric\lib\sax.jar;\electric\lib\servlet.jar;\electric\lib\xerces.jar;\electric\lib\jnet.jar;\electric\lib\GLUE-EXAMPLES.jar;\electric\lib\jsse.jar;
Add the GLUE jars to your classpath

- \electric\lib\GLUE-STD.jar;
- \electric\lib\dom.jar;
- \electric\lib\jsp.jar;
- \electric\lib\sax.jar;
- \electric\lib\servlet.jar;
- \electric\lib\xerces.jar;
Creating a webservice using Glue

• Create an interfaces that is the methods you want exosed as a web service
• Create a implementation class
• Create java class that has a main that call the publish() methods on the glue API
• Create a java class that invokes the webservice through the Glue API
Testing your GLUE implementation

- Open two command windos and go to the \electric\src\examples\publish directory
- Execute javac *.java
- In one window java Publish1
- In the other java Invoke1
A CreditBureauService that is published as a webservice

• I created the following files:
  – ICreditBureauService
  – CreditBureauService
  – PublishAndStartCreditBureauService
  – InvokeCreditBureauService
Invoking a webservice

Command Prompt - java PublishAndStartCreditBureauService
C:\Lectures\Lecture11>java PublishAndStartCreditBureauService
[STARTUP] GLUE standard 3.2.3 (c) 2001-2002 The Mind Electric
[DEPLOYMENT] publishing web service on path \system\application
[DEPLOYMENT] web service disabled. publish path \sample

Command Prompt
C:\Lectures\Lecture11>java InvokeCreditBureauService
Credit score by SSNum [123456789] = 123.0
Credit score by Name [Hafiz Ahsan] = 111.0
Credit score by Name through CBS2 [Hafiz Ahsan] = 111.0
Credit history by Name through CBS2 [Hafiz Ahsan] = <none>
Credit history by Name through CBS2 [Nobody Nobody] = <null>
Credit score by Name through CBS2 [ John Smith] = 123.0
Credit history = Bought a $300,000 house 1999.10.02
Credit score by Name through CBS2 [ Jane Smith] = 123.0
Credit history = Missed a loan payment 5 times 2000
Credit score by Name through CBS2 [ Isiu Chang] = 99.0
Credit history = Earned $250,000.00 in 2003
C:\Lectures\Lecture11>
If the service isn’t running here’s what the client does

```java
C:\Lectures\Lecture11>java InvokeCreditBureauService
Exception in thread "main" electric.registry.RegistryException: could not bind to path:
  services\CreditBureauService.wsdl1
at electric.soap.SOAPRegistry.bind(Unknown Source)
at electric.registry.compound.CompoundRegistry.bind(Unknown Source)
at electric.registry.Registry.getReference(Unknown Source)
at electric.registry.Registry.getProxy(Unknown Source)
at electric.registry.Registry.bind(Unknown Source)
at electric.registry.Registry.bind(Unknown Source)
at InvokeCreditBureauService.main(InvokeCreditBureauService.java:12)
```

Copyright Poelman & Associates, Inc. 2002 - All Rights Reserved
Web Service invoked via the servlet

Credit score by SSN [123456789] = 123.0
Credit score by Name [Hafiz Ahsan] = 111.0
Credit score by Name through CBS2 [Hafiz Ahsan] = 111.0
Credit history by Name through CBS2 [Hafiz Ahsan] =
Credit history by Name through CBS2 [Nobody Nobody] =
Credit score by Name through CBS2 [John Smith] = 123.0
Credit history = Bought a $300,000 house 1999.10.02
Credit score by Name through CBS2 [Jane Smith] = 123.0
Credit history = Missed a loan payment 5 times 2000 Credit score by Name through CBS2 [Issu Chang] = 99.0
Credit history = Earned $250,000.00 in 2003
Selected code from the servlet

```
import electric.registry.Registry;

protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, java.io.IOException {
    response.setContentType("text/html");
    java.io.PrintWriter out = response.getWriter();
    out.println("<html>");

    try {
        // bind to web service whose WSDL is at the specified URL
        String url = "http://localhost:8004/myWebServices/CreditBureauService.wsdl";
        ICreditBureauService creditBureauService =
            (ICreditBureauService) Registry.bind( url, CreditBureauService.class );

        LName = "Chang"; FName = "Isiu";
        creditScore = cBS2.getCreditScore(FName,LName);
        out.println( "Credit score by Name through CBS2 [ " + FName + " " + LName + "] = " + creditScore +"<br>");
        creditHistory = cBS2.getCreditHistory(FName,LName);
        out.println( "Credit history = " + creditHistory);
    }
    catch(Exception e)
    {
        out.println("Exception = " + e);
    }
```
Hints for getting a servlet to work with GLUE …

• Put the GLUE-STD.jar and other GLUE jars in the `\electric\lib` directory in the `lib` directory under your `WEB-INF` so the servlet engine can find them.
A note about GLUE

• GLUE automatically creates the WSDL data in memory for you. You don’t need to create a WSDL but you still should to see what it looks like.

• java2wsdl CreditBureauService -e http://localhost:8004/myWebServices
The WSDL created ...
The WSDL created ...(2)

```
<wsdl:portType name='CreditBureauService'>
  <wsdl:operation name='getCreditScore' parameterOrder='SSNumber'>
    <wsdl:input name='getCreditScore0In' message='tns:getCreditScore0In'/>
    <wsdl:output name='getCreditScore0Out' message='tns:getCreditScore0Out'/>
  </wsdl:operation>
  <wsdl:operation name='getCreditScore' parameterOrder='FName LName'>
    <wsdl:documentation>Return the score for an individual.</wsdl:documentation>
    <wsdl:input name='getCreditScore1In' message='tns:getCreditScore1In'/>
    <wsdl:output name='getCreditScore1Out' message='tns:getCreditScore1Out'/>
  </wsdl:operation>
  <wsdl:operation name='getCreditHistory' parameterOrder='SSNumber'>
    <wsdl:input name='getCreditHistory2In' message='tns:getCreditHistory2In'/>
    <wsdl:output name='getCreditHistory2Out' message='tns:getCreditHistory2Out'/>
  </wsdl:operation>
  <wsdl:operation name='getCreditHistory' parameterOrder='FName LName'>
    <wsdl:documentation>Return the history for an individual.</wsdl:documentation>
    <wsdl:input name='getCreditHistory3In' message='tns:getCreditHistory3In'/>
    <wsdl:output name='getCreditHistory3Out' message='tns:getCreditHistory3Out'/>
  </wsdl:operation>
</wsdl:portType>
```
The WSDL created ...(3)

```xml
<wSDL:operation name='getCreditScore'>
  <soap:operation soapAction='getCreditScore' style='rpc'/>
  <wsdl:input name='getCreditScore1In'>
    <soap:body use='encoded' namespace='http://tempuri.org/CreditBureauService'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding'/>
  </wsdl:input>
  <wsdl:output name='getCreditScore1Out'>
    <soap:body use='encoded' namespace='http://tempuri.org/CreditBureauService'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding'/>
  </wsdl:output>
</wSDL:operation>
```
Types of WS invocations

• RPC (Remote Procedure Call)
  – Client requests -> sends SOAP request via HTTP -> services gets SOAP request -> processes it and returns a SOAP response to client -> client receives the SOAP response and continues execution.
  – Blocks waiting for a response
  – Not very scalable
Types of WS invocations

• Doc (document)
  – Client requests -> sends SOAP request via HTTP and doesn’t block.
  – Client also has a portion of code that acts as a WS server for responses. It waits for a response to come in and processes it async.
  – More scalable
  – Harder to get through a firewall because clients and servers exist on both sides of the firewall.
WS info

• A set of web standards currently
• Not all vendor implementations are interoperable!
• Over hyped!!!
• Not a replacement within apps for RMI/COM+/CORBA
• No management technology of WS (unlike EJBs yet in place) Need a Web Service Server! Or WS Manager.
• WS are really interfaces to services hosted as EJBs, Servlets, java processes, COM objects, etc.
• Most EJB servers have added WS features
• Not required to be HTTP!
  – JMS, Email (SMTP), TCP, ?
www.grandcentral.com

a site for webservices integration and security, ...
XMethods (www.xmethods.com)
a web site devoted to publicly published web services
Current Limitations of WS

- Very limited security capabilities
- Non-transactional – HTTP can’t be transactional by its very nature
- Waiting for a “killer app”
- Doc version and firewall configurations are still incompatible
- Not many really understand the value and power – most sending objects over WS, should just send data!
- HTTP only
A thought I had …

• As part of your final exam …
• You would write a java WS client that invokes a WS in my office to send me your final (as a java object).
• Then later to get your grade you’d need to write a different client to talk to a grade reporting WS!!
• But then I came back to reality.
• I will NOT require this! (or even attempt it ;-)
The SUN WebServices packages

- You must have a JSK that is >= 1.3.1!
- It's >30 megs compressed and expand to 59+ megs!
WSDP is …

• JAXM and SAAJ
• JAXP
• JAXR
• JAX-RPC
• JSTL
• Tomcat
• Ant
• Registry Server
Summary

• WebService =
  – HTTP
  – SOAP (which is XML)
  – WSDL (which is XML)
  – UDDI (which is a webservice based interface to a directory of web services)

• Gets through firewalls
• RPC vs DOC
• Independent of platform and dev language!
• Non transactional and weak security
• Web Services are “da bomb diggity” of technology right now! (Very over hyped)
resources

• WebServices Blueprints
  • http://java.sun.com/blueprints/webservices/using/webservbp.html
  • http://java.sun.com/webservices/webservicespack.html
  • http://www.soapclient.com/uddisearch.html
  • http://www.xmethods.com/