Web Services Overview

Marlon Pierce
Community Grids Lab
Indiana University

Assignments

- Download and install Tomcat (again).
  - http://jakarta.apache.org/tomcat/
  - You will need two tomcat servers.
- Install Apache Axis.
  - Use “HappyAxis” to make sure you have done so correctly.
  - http://ws.apache.org/axis/
- Design and deploy a sample web service.
- Write a client application to use the web service.
- Use Google and Amazon WSDL to design your own client.

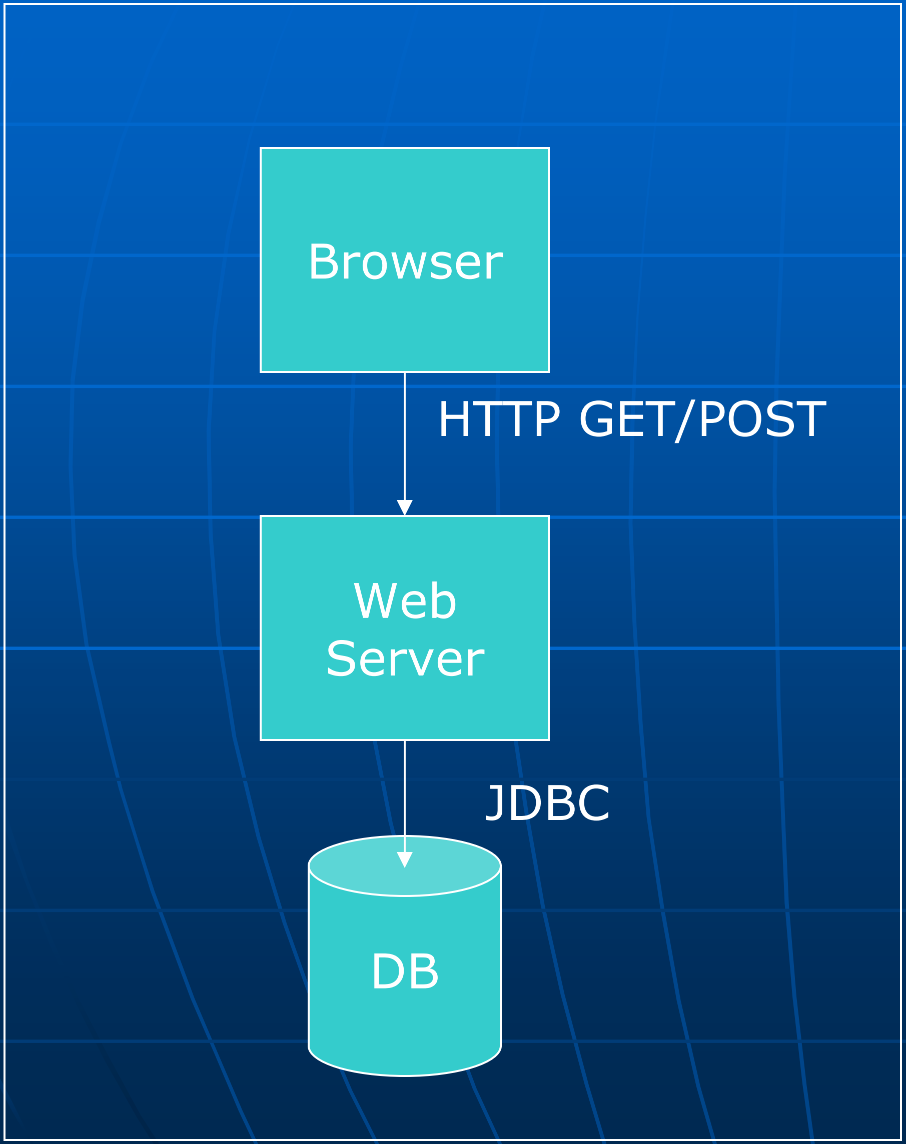
This Lecture…

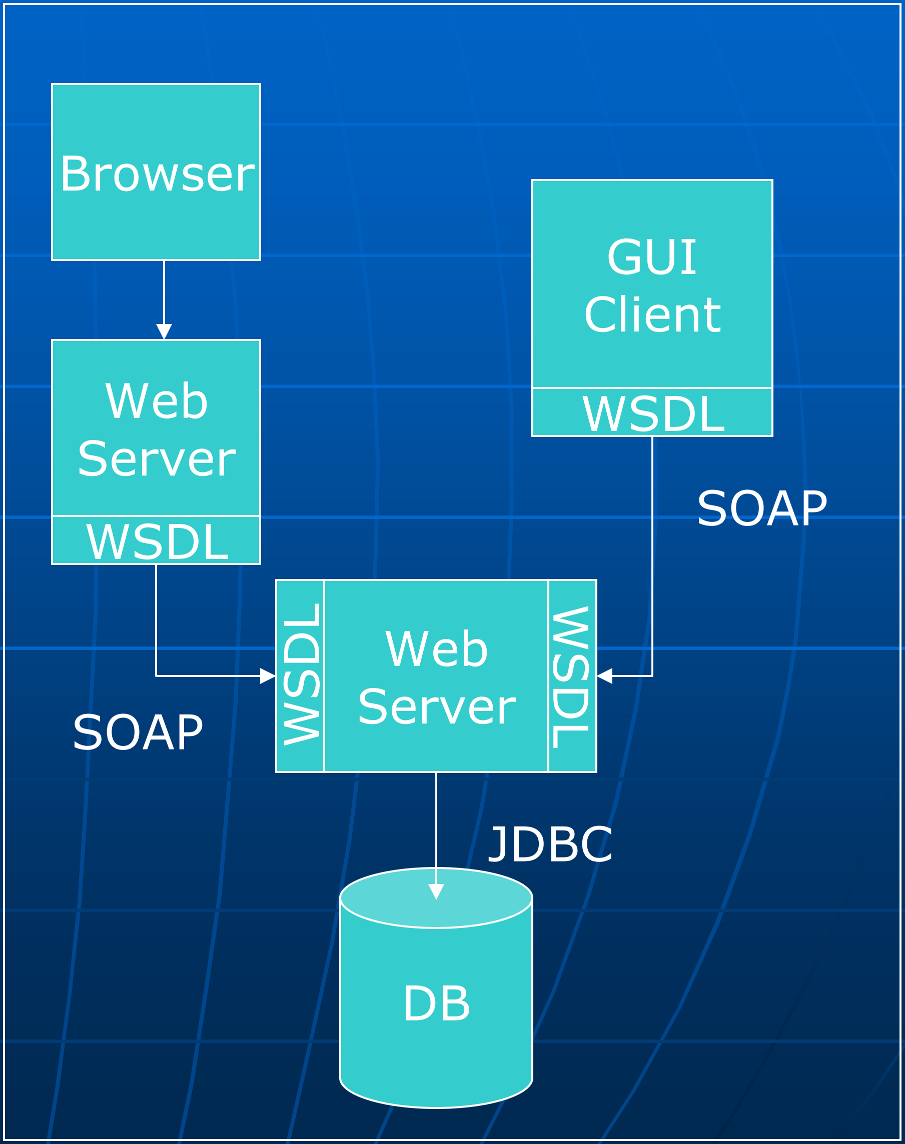
- This lecture is intended to introduce the main concepts of Web Services.
- We will also look at some things (SOAP, WSDL) in detail…
- But the primary purpose is to introduce topics that will all be covered in greater detail in future lectures.

What Are Web Services?

- Web services framework is an XML-based distributed object/service/component system.
  - SOAP, WSDL, WSIL, UDDI
  - Intended to support machine-to-machine interactions over the network.
- Basic ideas is to build an platform and programming language-independent distributed invocation system out of existing Web standards.
  - Most standards defined by W3C, Oasis (IP considerations)
  - Interoperability really works, as long as you can map XML message to a programming language type, structure, class, etc.
- Very loosely defined, when compared to CORBA, etc.
- Inherit both good and bad of the web
  - Scalable, simple, distributed
  - But no centralized management, system is inefficient, must be tolerant of failures.

Basic Architectures: Servlets/CGI and Web Services





Explanation of Previous Slide

- The diagram on the left represents a standard web application.
  - Browsers converse with web servers using HTTP GET/POST methods.
  - Servlets or CGI scripts process the parameters and take action, like connect to a DB.
  - Examples: Google, Amazon
- On the right, we have a Web services system.
  - Interactions may be either through the browser or through a desktop client (Java Swing, Python, Windows, etc.)
  - I will explain how to do this in several more lectures.
  - Examples: Google, Amazon

Some Terminology

- The diagram on the left is called a client/server system.
- The diagram on the right is called a multi-tiered architecture.
- SOAP: Simple Object Access Protocol
  - XML Message format between client and service.
- WSDL: Web Service Description Language.
  - Describes how the service is to be used
  - Compare (for example) to Java Interface.
  - Guideline for constructing SOAP messages.
  - WSDL is an XML language for writing Application Programmer Interfaces (APIs).

Amazon and Google Experiment with Web Services

- Both Google and Amazon have conducted open experiments with Web services.
- Why? To allow partners to develop custom user interfaces and applications that work Google and Amazon data and services.
- You can download their APIs and try them.
  - http://www.google.com/apis/
  - http://www.amazon.com/webservices

Why Use Web Services?

- Web services provide a clean separation between a capability and its user interface.
- This allows a company (Google) with a sophisticated capability and huge amounts of data to make that capability available to its partners.
  - "Don't worry about how PageRank works or web robots or data storage. We will do that. You just use this WSDL API to build your client application to use our search engine."

A Google Aside

- Google's PageRank system was developed by two Stanford grad students.
- Open algorithm published in scholarly journals, conferences.
  - Previous (and lousy) search engines were all proprietary.
- See for example http://www7.scu.edu.au/programme/fullpapers/1921/com1921.htm

When To Use Web Services?

- Applications do not have severe restrictions on reliability and speed.
- Two or more organizations need to cooperate
  - One needs to write an application that uses another's service.
- Services can be upgraded independently of clients.
  - Google can improve PageRank implemenation without telling me.
  - Just don't change the WSDL.
- Services can be easily expressed with simple request/response semantics and simple state.
  - HTTP and Cookies, for example.

Relationship to Previous Work

Connecting to Bryan’s Lectures on XML, Java, Java Servlets and JSP.

XML Overview

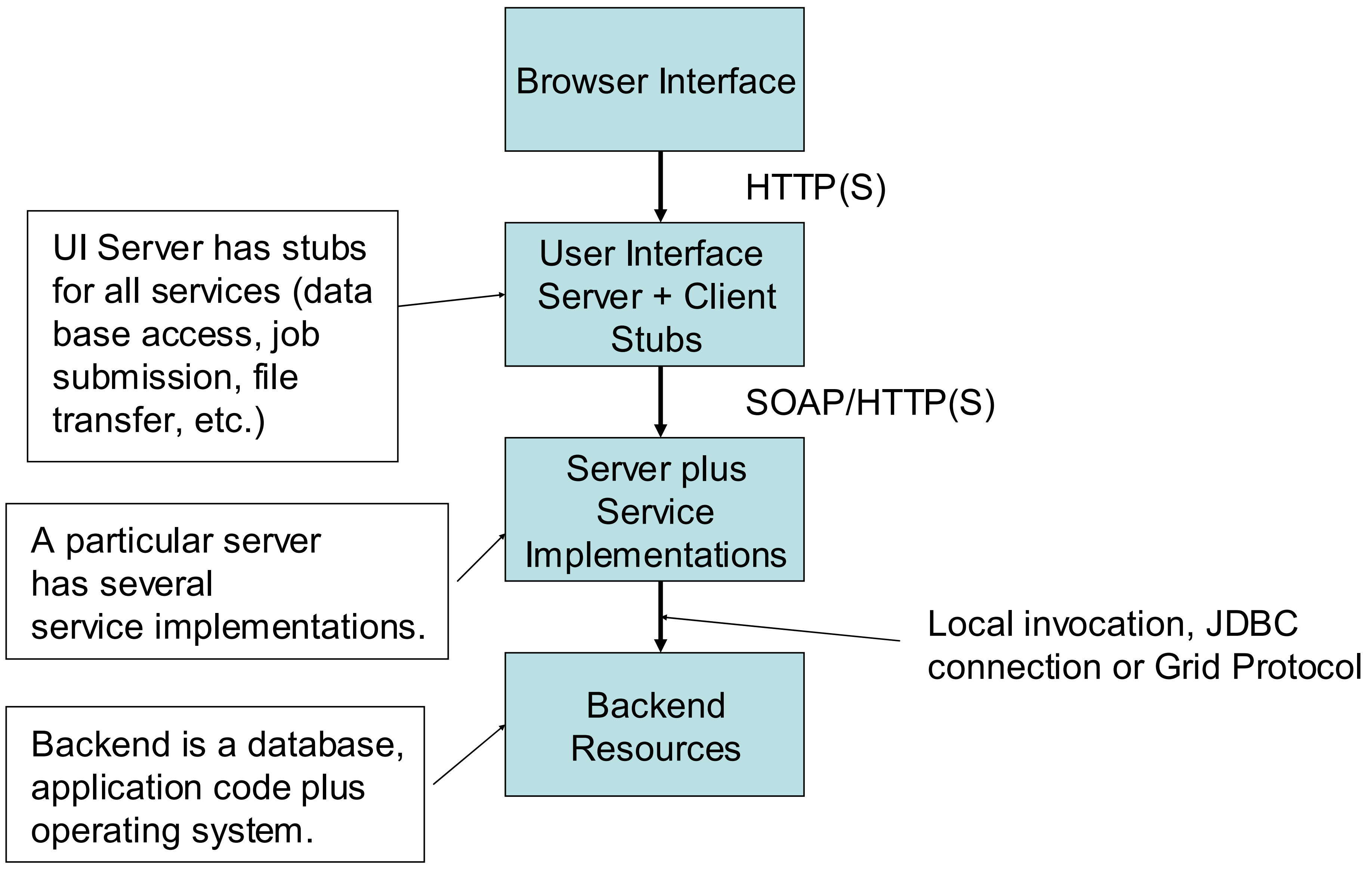
- XML is a language for building languages.
- Basic rules: be well formed and be valid
- Particular XML "dialects" are defined by an XML Schema.
  - XML itself is defined by its own schema.
- XML is extensible via namespaces
- Many non-Web services dialects
  - RDF, SVG,GML, XForms, XHTML
- Many basic tools available: parsers, XPath and XQuery for searching/querying, etc.

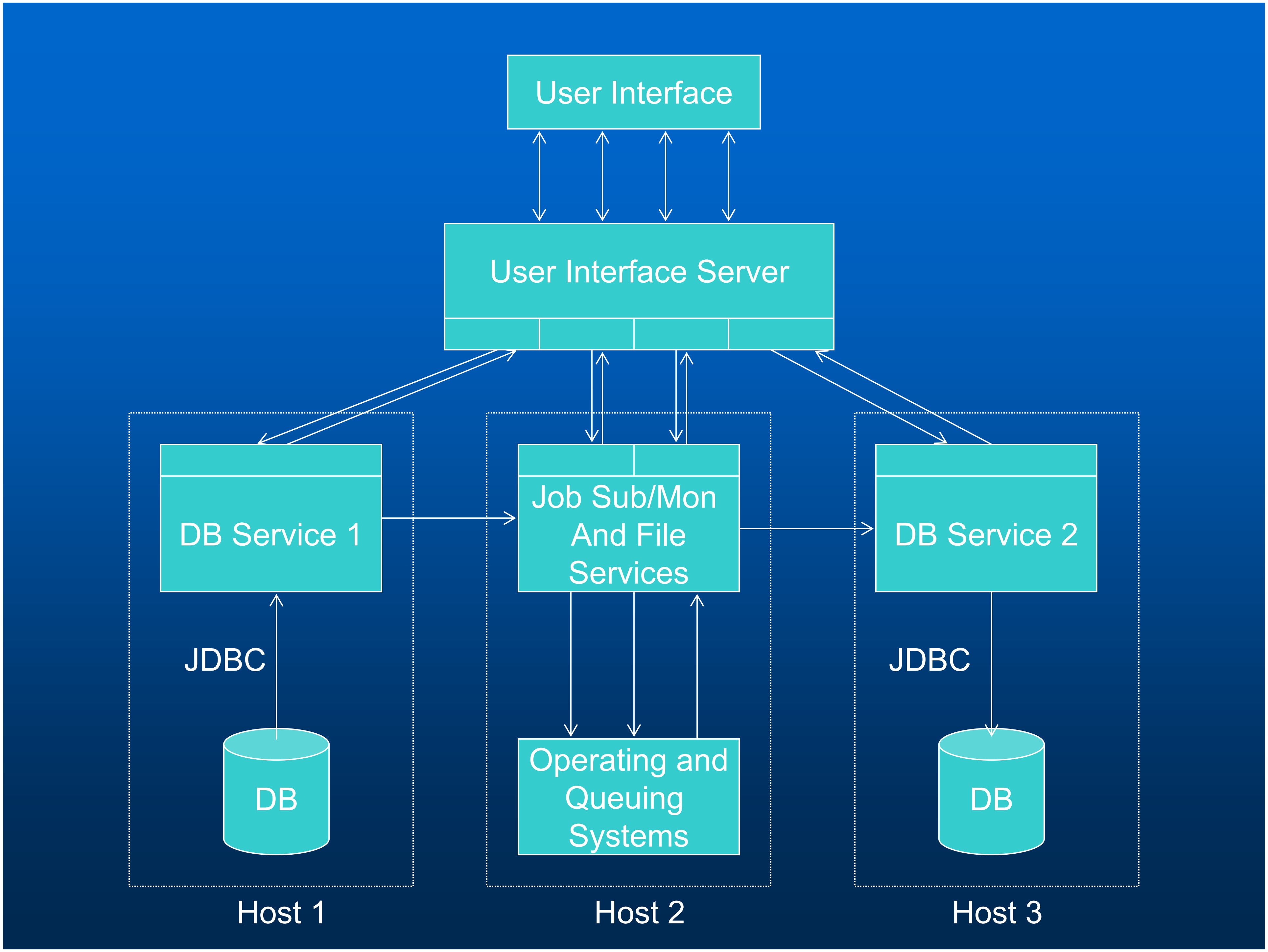
XML and Web services

- XML provides a natural substrate for distributed computing:
  - Its just a data description.
  - Platform, programming language independent.
- So let’s describe the pieces.
- Web Services Description Language (WSDL)
  - Describes how to invoke a service (compare with CORBA IDL).
  - Can bind to SOAP, other protocols for actual invocation.
- Simple Object Access Protocol (SOAP)
  - Wire protocol extension for conveying RPC calls.
  - Can be carried over HTTP, SMTP.

Web Service Architectures

- The following examples illustrate how Web services interact with clients.
- For us, a client is typically a JSP, servlet, or portlet that a user accesses through browser.
- You can also build other clients
  - Web service interoperability means that clients and services can be in different programming languages (C/C++, python, java, etc).


Browser Interface
HTTP(S)
User Interface Server + Client Stubs
SOAP/HTTP(S)
Server plus Service Implementations
Local invocation, JDBC connection or Grid Protocol
Backend Resources
UI Server has stubs for all services (data base access, job submission, file transfer, etc.)
A particular server has several service implementations.
Backend is a database, application code plus operating system.


User Interface
User Interface Server
DB Service 1
Job Sub/Mon And File Services
DB Service 2
JDBC
DB
Operating and Queuing Systems
JDBC
DB
Host 1
Host 2
Host 3

Before Going On…

- In the next several slides we'll go into the details of WSDL and SOAP.
- But in practice, you don't need to work directly with either.
  - Most tools that I'm familiar with generate the WSDL for you from your class.
  - SOAP messages are constructed by classes.
  - Generated client stubs will even hide SOAP classes behind a local "façade" that looks like a local class but actually constructs SOAP calls to the remote server.

Web Services Description Language

Defines what your service does and how it is invoked.

WSDL Overview

- WSDL is an XML-based Interface Definition Language.
  - You can define the APIs for all of your services in WSDL.
- WSDL docs are broken into five major parts:
  - Data definitions (in XML) for custom types
  - Abstract message definitions (request, response)
  - Organization of messages into “ports” and “operations” (→classes and methods).
  - Protocol bindings (to SOAP, for example)
  - Service point locations (URLs)
- Some interesting features
  - A single WSDL document can describe several versions of an interface.
  - A single WSDL doc can describe several related services.

The Java Code

```
public String[] execLocalCommand(String
   command) {
    Runtime rt = Runtime.getRuntime();
    String stdout="",stderr="";
    try {
      Process p = rt.exec(command);
      BufferedReader in=
        new BufferedReader(new
        InputStreamReader(p.getInputStream()));
      BufferedReader err=
        new BufferedReader(new
        InputStreamReader(p.getErrorStream()));
```

Java Code Continued

```
   String line;
    while((line=in.readLine())!= null)
        {stdout+=line+"\n";}
    in.close();
    while ((line=err.readLine())!=null)
        {stderr+=line+"\n";}
    err.close();
 }//End of try{}
 catch (Exception eio) {…}
 String[] retstring=new String[2];
    retstring[0]=stdout;
    retstring[1]=stderr;
    return retstring;
} //End of method
```

WSDL Example: Job Submission

- Our example is a simple service that can executes local (to the server) commands.
- Service implementation (in Java) has a single method
  - ExecLocal takes a single string argument (the command to exec)
  - Returns a 2D string array (standard out and error).
- The WSDL maps to a Java interface in this case.

The Full WSDL

- The following slide contains the WSDL definition for the Job Submit service.
  - I omitted some data definitions to get into one page with a decent font.
- As you can see, WSDL is very verbose
  - Typically, you don’t write WSDL
  - This file was actually generated from my Java class by Apache Axis.
- We will go through the parts of the doc in some detail.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions>
<wsdl:message name="execLocalCommandResponse">
<wsdl:message name="execLocalCommandRequest">
<wsdl:portType name="SJwsImp">
<wsdl:operation name="execLocalCommand" parameterOrder="in0">
      <wsdl:input message="impl:execLocalCommandRequest"
name="execLocalCommandRequest"/>
      <wsdl:output message="impl:execLocalCommandResponse"
name="execLocalCommandResponse"/>
 </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="SubmitjobSoapBinding" type="impl:SJwsImp">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="execLocalCommand">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="execLocalCommandRequest">
      <wsdl:output name="execLocalCommandResponse">
</wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="SJwsImpService">
    <wsdl:port binding="impl:SubmitjobSoapBinding" name="Submitjob">
  </wsdl:service>
</wsdl:definitions>
```

WSDL Elements I

- Types: describes custom XML data types (optional) used in messages.
  - For OO languages, types are a limited object serialization.
  - We'll see an example for defining arrays.
- Message: abstractly defines the messages that need to be exchanged.
  - Conventionally messages are used to group requests and responses.
  - Each method/function in the interface contains 0-1 request and 0-1 response messages.
  - Consists of *part* elements. Usually you need one part for each variable sent or received. Parts can either be XML primitive types or custom complex types.

Types for Job Submission

- Recall that the job submission service sends a string (the command) and returns a 2D array.
- Strings are XML Schema primitive types, so we don't need a special definition in our WSDL.
- Arrays are not primitive types. They are defined in the SOAP schema, so we will import that definition.
  - In other words, SOAP has rules for array encoding; vanilla XML does not.

Example: WSDL *types* for Custom Data Definition

```
<wsdl:types>
  <schema targetNamespace="http://.../GCWS/services/Submitjob"
                    xmlns:impl="http://.../GCWS/services/Submitjob"
                    xmlns="http://www.w3.org/2001/XMLSchema">
     <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
     <complexType name="ArrayOf_xsd_string">
          <complexContent>
            <restriction base="soapenc:Array">
               <attribute ref="soapenc:arrayType"
                            wsdl:arrayType="xsd:string[]" />
            </restriction>
          </complexContent>
      </complexType>
      <element name="ArrayOf_xsd_string" nillable="true"
                         type="impl:ArrayOf_xsd_string" />
  </schema>
</wsdl:types>
```

What Does It Mean?

- We start with some useful namespace definitions.
- We next import the SOAP schema
  - It has the array definitions we need.
- Finally, we define our own local XML complex type, ArrayOf_xsd_string.
  - This extends the SOAP array type
  - We restrict this to String arrays.

Message Elements for Job Submission Service

- Our service implementation has one method of the form (in Java)

  public String[] execLocalCommand(String cmd)
- This will require one “request” message and one “response” message.
- Each message has one *part:*
  - Request message must send the String cmd.
  - Response must get back the String[] array (defined previously as a custom type).
- If we had to pass two input variables, our “request” message would need two part elements.
- Note the name attributes of messages are important!

Message Examples for Job Submission Service

```
<wsdl:message
        name="execLocalCommandResponse">
    <wsdl:part
        name="execLocalCommandReturn"
        type="impl:ArrayOf_xsd_string" />
</wsdl:message>
<wsdl:message
        name="execLocalCommandRequest">
    <wsdl:part name="in0" type="xsd:string" />
</wsdl:message>
```

portTypes

- *portType* elements map messages to *operations*.
- You can think of portType==class, operation==class methods.
- Operations can contain input, output, and/or fault bindings for messages.
- An operation may support of the following message styles:
  - One-way: request only
  - Two-way: request/response
  - Solicit-response: server “push” and client response
  - Notification: one-way server push

portType for JobSubmit

- We previously defined the messages and types needed. Now we bind them into the portType structure.
- PortType names are important
  - Will be referenced by *binding* element.
- Note names of previously defined messages are used as references in the operations.

Example WSDL Nugget

```
<wsdl:portType name="SJwsImp">
  <wsdl:operation name="execLocalCommand"
                              parameterOrder="in0">
    <wsdl:input
        message="impl:execLocalCommandRequest"
        name="execLocalCommandRequest" />
    <wsdl:output
        message="impl:execLocalCommandResponse"
        name="execLocalCommandResponse" />
  </wsdl:operation>
</wsdl:portType>
```

Some Notes on the PortType Definition

- PortTypes refer to messages by name
  - The message attribute in <input> and <output> elements of <operation> refer to the name attributes of the previously defined messages.
  - The operation and portType names will similarly be used for reference in forthcoming tags.
- Also note "parameterOrder" does what you would expect. For the current example, there is only one input parameter.

PortType Bindings

- portTypes are abstract interface definitions.
  - Don’t say anything about how to invoke a remote method.
- Remote invocations are defined in *binding* elements.
- Binding elements are really just place holders that are extended for specific protocols
  - WSDL spec provides SOAP, HTTP GET/POST, and MIME extension schema examples.

SOAP Bindings for JobSubmit Service

- Note that the binding element contains a mixture of tags from different namespaces (wsdl and wsdlsoap).
- WSDL child elements for *binding* element are *operation, input,* and *output.*
- WSDLSOAP elements are from a different XML schema (a new one, neither WSDL nor SOAP).
    - This is how you extend WSDL bindings: define a new schema that gives mapping instructions from WSDL to the protocol of choice.
- The binding element name is important, will be used as a reference by the final port binding.

```
<wsdl:binding
          name="SubmitjobSoapBinding" type="impl:SJwsImp">
  <wsdlsoap:binding style="rpc"
          transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="execLocalCommand">
       <wsdlsoap:operation soapAction="" />
         <wsdl:input name="execLocalCommandRequest">
            <wsdlsoap:body
                encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
                namespace="http://.../GCWS/services/Submitjob"
                use="encoded" />
         </wsdl:input>
         <wsdl:output name="execLocalCommandResponse">
            <wsdlsoap:body
                encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
                namespace=http://.../GCWS/services/Submitjob
                use="encoded" />
         </wsdl:output>
     </wsdl:operation>
 </wsdl:binding>
```

A Closer Look at SOAP Binding

```
<wsdlsoap:body
   encodingStyle=http://schemas.xmlsoap.org/
   soap/encoding/
   namespace=http://.../GCWS/services/Submi
   tjob use="encoded" />
```

- All this really means is "encode the message by the rules in encodingStyle and put it in the SOAP body."
- The bindings are just instructions that must be implemented by the SOAP message generator.

Service and Port Definitions

- So far, we have defined the class method interfaces (portTypes) and the rules for binding to a particular protocol.
- *Port* elements define how the bindings (and thus the portTypes) are associated with a particular server.
- The *service* element collects *ports.*

Service and Port Elements for the Job Submission Service

```
<wsdl:service name="SJwsImpService">
  <wsdl:port
       binding="impl:SubmitjobSoapBinding"
       name="Submitjob">
     <wsdlsoap:address
       location="http://.../GCWS/services/Submitjob" />
  </wsdl:port>
</wsdl:service>
```

Explanation

- Note the port element’s binding attribute points to the appropriate *binding* element by name.
- The only purpose of the port element is to point to a service location (a URL). This is done by extension (SOAP in this case.)
- Ports are child elements of the *service* element. A service can contain one or more ports.
  - Note the value of multiple ports: a single portType may correspond to several ports, each with a different protocol binding and service point.

WSDL Trivia

- The schema rules allow all of the elements we have discussed to appear zero or more times.
- A single WSDL file may contain many portTypes (although this is not usual).
  - You may want to do this to support multiple interface definitions of a service for backward compatibility.
- Multiple ports may also be used to provide different views of a service
  - One portType defines the interface.
  - Another provides access to metadata about the service.
  - Yet another may define how the service interacts with other services via notification/event systems.

Simple Object Access Protocol

A message format for exchanging structured, typed information

SOAP Basics

- SOAP is often thought of as a protocol extension for doing RPC over HTTP.
- This is not completely accurate: SOAP is an XML message format for exchanging structured, typed data.
- It may be used for RPC in client-server applications but is also suitable for messaging systems (like JMS) that follow one-to-many (or publish-subscribe) models.
- SOAP is not a transport protocol. You must attach your message to a transport mechanism like HTTP.

SOAP Structure

- A SOAP message is contained in an *envelop.*
- The envelop element in turn contain (in order)
  - An optional *header* with one or more child entries.
  - A *body* element that can contain one or more child entries. These child entries may contain arbitrary XML data.

SOAP Headers

- Headers are really just extension points where you can include elements from other namespaces.
  - i.e., headers can contain arbitrary XML.
- Header entries may optionally have a “mustUnderstand” attribute.
  - mustUnderstand=1 means the message recipient must process the header element.
  - If mustUnderstand=0 or is missing, the header element is optional.

SOAP Body

- Body entries are really just placeholders for arbitrary XML from some other namespace.
- The body contains the XML message that you are transmitting.
- The message format is not specified by SOAP.
  - The <Body></Body> tag pairs are just a way to notify the recipient that the actual XML message is contained therein.
  - The recipient decides what to do with the message.

Example Messages

- Recall the WSDL interface for “SubmitJob”
  - Sends one string command
  - Returns array of strings for standard out and error.
- The envelop is decorated with a few useful namespaces:
  - soapenv defines the version
  - xsd is the Schema definition itself
  - xsi defines some useful constants.
- The body is just an arbitrary XML fragment.
  - Assumes the recipient knows what this means.
  - Recipient must looks up the ExecLocalCommand operation in the JobSubmit service and passes it one string argument.
  - The ns1 namespace tells the recipient the WSDL namespace that defines the service.
  - xsi:type lets the recipient know that the arbitrary XML element in0 is in fact a string, as defined by the XML Schema.

SOAP Request

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd=http://www.w3.org/2001/XMLSchema
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:execLocalCommand
          soapenv:encodingStyle
            ="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:ns1
          ="http://.../GCWS/services/Submitjob/GCWS/services/Submitjob">
        <in0 xsi:type="xsd:string">/usr/bin/csh /tmp/job.script</in0>
    </ns1:execLocalCommand>
  </soapenv:Body>
</soapenv:Envelope>
```

Example Response

- The structure is the same as the request.
- The interesting thing here is that the request returns a 2-element array of two strings.
    - Arrays not defined by XML schema
    - SOAP encoding does define arrays, so use xsi:type to point to this definition.
    - <item></item> surrounds each array element.
- Note that arbitrary XML returns can likewise be encoded this way.
    - Use xsi:type to point to a schema.

SOAP Response

```
<soapenv:Envelope
            xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
            xmlns:xsd=http://www.w3.org/2001/XMLSchema
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
    <ns1:execLocalCommandResponse
          soapenv:encodingStyle=
             http://schemas.xmlsoap.org/soap/encoding/
          xmlns:ns1="http://../services/Submitjob">
       <execLocalCommandReturn xsi:type="soapenc:Array“
            soapenc:arrayType="xsd:string[2]"
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
        <item></item>    <item></item>
        </execLocalCommandReturn>
    </ns1:execLocalCommandResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Developing Web Services

Using Apache Axis to develop Java implementations of Web services.

Web Service Development Tools

- Web service toolkits exist for various programming languages:
  - C++,Python, Perl, various Microsoft .NET kits.
- We’ll concentrate on building Java Web services with Apache Axis.
- Language and implementation interoperability is addressed through WS-I.
  - http://www.ws-i.org/

Apache Axis Overview

- Apache Axis is a toolkit for converting Java applications into Web services.
- Axis service deployment tools allow you to publish your service in a particular application server (Tomcat).
- Axis client tools allow you to convert WSDL into client stubs.
- Axis runtime tools accept incoming SOAP requests and redirect them to the appropriate service.

Developing and Deploying a Service

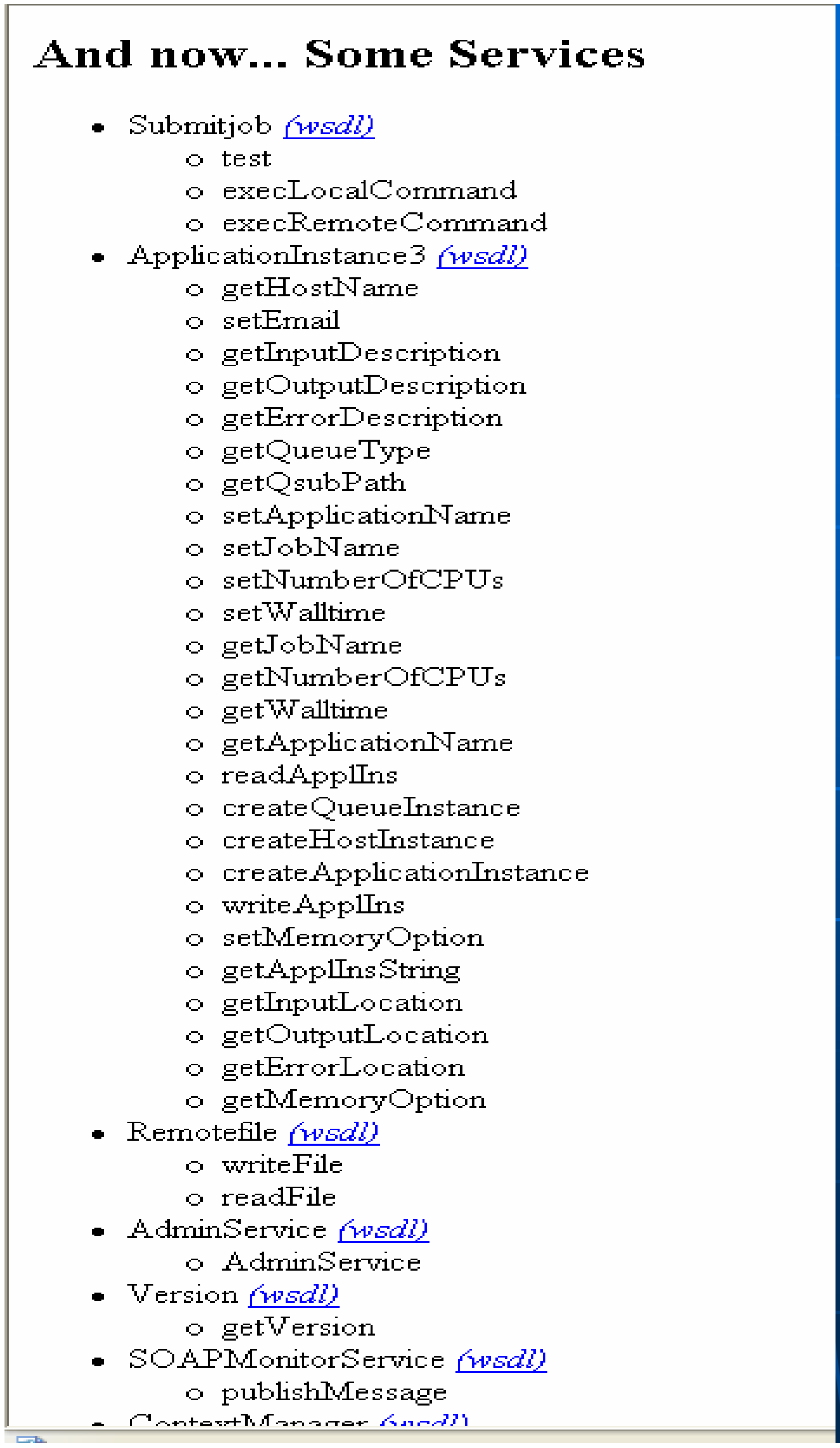
- Download and install Tomcat and Axis.
- Write a Java implementation
  - Our SubmitJob is a simple example but services can get quite complicated.
  - Compile it into Tomcat’s classpath.
- Write a deployment descriptor (WSDD) for your service.
  - Will be used by Axis runtime to direct SOAP calls.
- Use Axis’s AdminClient tool to install your WSDD file.
  - The tells the axis servlet to load your class and direct SOAP requests to it.
- That’s it.
  - Axis will automatically generate the WSDL for your service.

Sample WSDD

```
<deployment name="Submitjob"
      xmlns="http://xml.apache.org/axis/wsdd/"
      xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
      <service name="Submitjob" provider="java:RPC">
            <parameter name="scope" value="request"/>
            <parameter name="className"
                        value="WebFlowSoap.SJwsImp"/>
            <parameter name="allowedMethods"
                        value="execLocalCommand"/>
      </service>
</deployment>
```

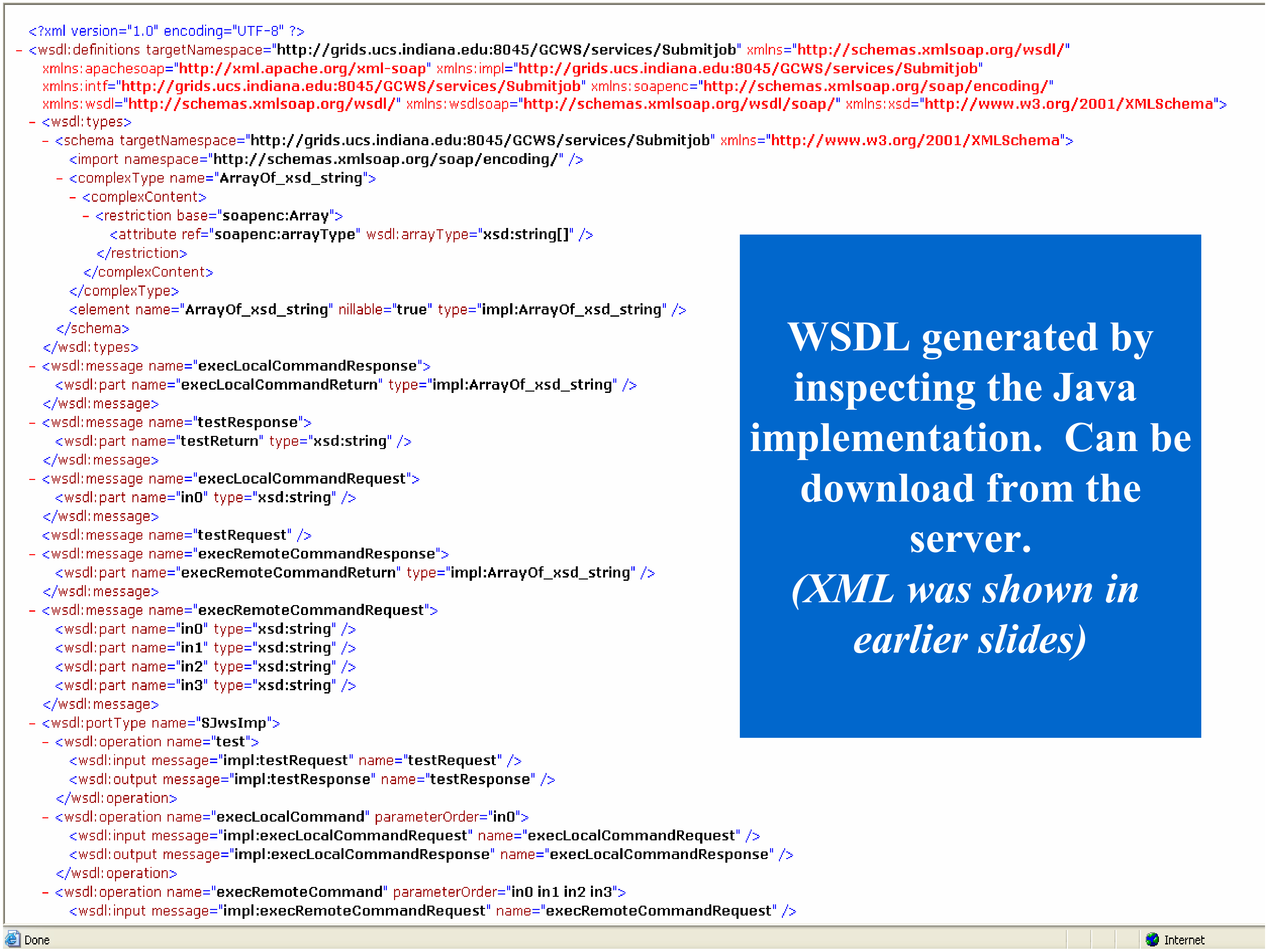
Explanation

- Use Axis's command-line AdminClient tool to deploy this to the server.
- Axis will create a service called
  - http://your.server/services/SubmitJob
- WSDL for service is available from
  - http://your.server/services/SubmitJob?wsdl
- A list of all services is available from
  - http://your.server/services


And now... Some Services

- Submitjob (wsdl)
  - test
  - execLocalCommand
  - execRemoteCommand
- ApplicationInstance3 (wsdl)
  - getHostName
  - setEmail
  - getInputDescription
  - getOutputDescription
  - getErrorDescription
  - getQueueType
  - getQsubPath
  - setApplicationName
  - setJobName
  - setNumberOfCPUs
  - setWalltime
  - getJobName
  - getNumberOfCPUs
  - getWalltime
  - getApplicationName
  - readApplIns
  - createQueueInstance
  - createHostInstance
  - createApplicationInstance
  - writeApplIns
  - setMemoryOption
  - getApplInsString
  - getInputLocation
  - getOutputLocation
  - getErrorLocation
  - getMemoryOption
- Remotefile (wsdl)
  - writeFile
  - readFile
- AdminService (wsdl)
  - AdminService
- Version (wsdl)
  - getVersion
- SOAPMonitorService (wsdl)
  - publishMessage
- ContextManager (wsdl)

Check your Tomcat Server for a list of deployed services.



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://grids.ucs.indiana.edu:8045/GCWS/services/Submitjob" xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://grids.ucs.indiana.edu:8045/GCWS/services/Submitjob"
    xmlns:intf="http://grids.ucs.indiana.edu:8045/GCWS/services/Submitjob" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <wsdl:types>
    - <schema targetNamespace="http://grids.ucs.indiana.edu:8045/GCWS/services/Submitjob" xmlns="http://www.w3.org/2001/XMLSchema">
        <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      - <complexType name="ArrayOf_xsd_string">
        - <complexContent>
          - <restriction base="soapenc:Array">
              <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]" />
            </restriction>
          </complexContent>
        </complexType>
        <element name="ArrayOf_xsd_string" nillable="true" type="impl:ArrayOf_xsd_string" />
      </schema>
    </wsdl:types>
  - <wsdl:message name="execLocalCommandResponse">
      <wsdl:part name="execLocalCommandReturn" type="impl:ArrayOf_xsd_string" />
    </wsdl:message>
  - <wsdl:message name="testResponse">
      <wsdl:part name="testReturn" type="xsd:string" />
    </wsdl:message>
  - <wsdl:message name="execLocalCommandRequest">
      <wsdl:part name="in0" type="xsd:string" />
    </wsdl:message>
    <wsdl:message name="testRequest" />
  - <wsdl:message name="execRemoteCommandResponse">
      <wsdl:part name="execRemoteCommandReturn" type="impl:ArrayOf_xsd_string" />
    </wsdl:message>
  - <wsdl:message name="execRemoteCommandRequest">
      <wsdl:part name="in0" type="xsd:string" />
      <wsdl:part name="in1" type="xsd:string" />
      <wsdl:part name="in2" type="xsd:string" />
      <wsdl:part name="in3" type="xsd:string" />
    </wsdl:message>
  - <wsdl:portType name="SJwsImp">
    - <wsdl:operation name="test">
        <wsdl:input message="impl:testRequest" name="testRequest" />
        <wsdl:output message="impl:testResponse" name="testResponse" />
      </wsdl:operation>
    - <wsdl:operation name="execLocalCommand" parameterOrder="in0">
        <wsdl:input message="impl:execLocalCommandRequest" name="execLocalCommandRequest" />
        <wsdl:output message="impl:execLocalCommandResponse" name="execLocalCommandResponse" />
      </wsdl:operation>
    - <wsdl:operation name="execRemoteCommand" parameterOrder="in0 in1 in2 in3">
        <wsdl:input message="impl:execRemoteCommandRequest" name="execRemoteCommandRequest" />
```

WSDL generated by inspecting the Java implementation. Can be download from the server.

(XML was shown in earlier slides)

Building a Client with Axis

- Obtain the WSDL file.
- Generate client stubs
  - Stubs look like local objects but really convert method invocations into SOAP calls.
- Write a client application with the stubs
  - Can be a Java GUI, a JSP page, etc.
- Compile everything and run.

Sample Java Client Code

```
/**Create SubmitJob client object and point to the
   service you want to use */
SubmiJob sjws = new
   SubmitJobServiceLocator().getSubmitjob(new

   URL(http://your.server/services/SubmitJob));
/** Invoke the method as if local. */
String[] messages =
        sjws.execLocalCommand(command);
```

Two Notes On Client Stubs

- Axis stubs convert method calls into SOAP requests but WSDL does not require the use of SOAP.
    - Web Service Invocation Framework (WSIF) from IBM allows flexibility of protocols. (Alek Slominski, IU)
- Client stubs introduce versioning problems.
    - We are developing dynamic (stubless) clients that construct SOAP messages by inspecting WSDL at runtime.

Web Service URLs

- Java
  - http://xml.apache.org/axis/
- XSOAP: C++ and Java toolkits for WS
  - http://www.extreme.indiana.edu/xgws/xsoap/
- gSOAP: C++ SOAP toolkit
  - http://www.cs.fsu.edu/~engelen/soap.html
- Python Web Services:
  - http://pywebsvcs.sourceforge.net/
- Perl:
  - http://www.soaplite.com/