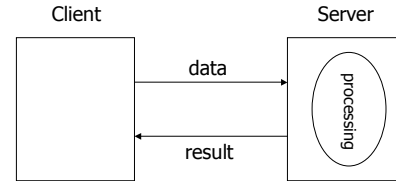


# CS520 Web Programming

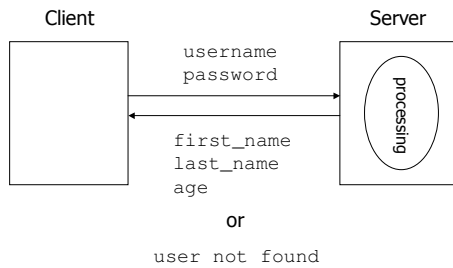
## Introduction to Web Services

Chengyu Sun  
California State University, Los Angeles

## Client-Server Architecture



## Client-Server Example

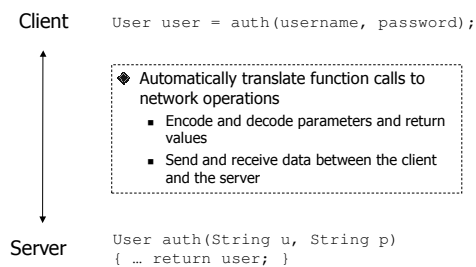


## Socket Programming – Client

```
create socket
write string to socket
write string to socket
read string from socket
if ( "user not found" ) return null;
else
    return new User (
        read string from socket
        read string from socket
        read integer from socket
    )
close socket
```

- ◆ Tedious networking code
- ◆ Application specific data exchange protocols

## Client-Server Interaction as Function Calls



## RPC and RMI

- ◆ Remote Procedure Call (RPC)
  - C
- ◆ Remote Method Invocation (RMI)
  - Java

## RMI – Server

- ◆ Create a service interface
  - Remote interface
  - Declares the methods to be remotely invoked
- ◆ Create a service implementation
  - Remote object
  - Implements the methods to be remotely invoked
- ◆ Register the service with a RMI registry so a client can find and use this service

## RMI – Client

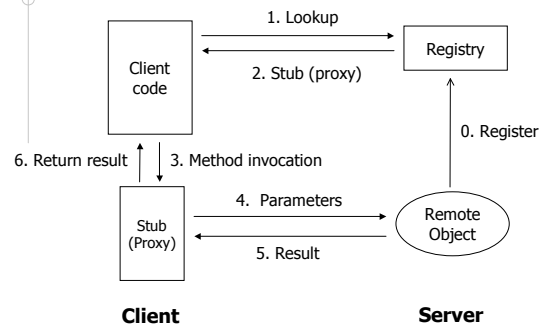
- ◆ Connect to the RMI registry
- ◆ Look up the service by name
- ◆ Invoke the service

## RMI Example – AuthService

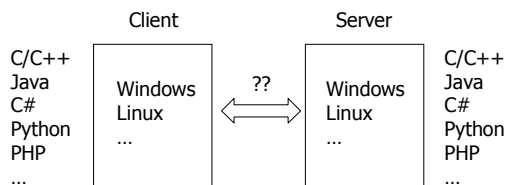
- ◆ Shared by both server and client
  - AuthService
  - User
- ◆ Server
  - AuthServiceImpl
  - AuthServiceStartup
- ◆ Client
  - AuthServiceClient

*Why does User have to implement the Serializable interface?  
What exactly does registry.lookup() return?*

## How RMI Works



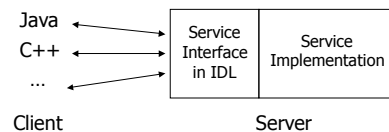
## Cross Platform RPC



- ◆ The client and the server use different languages and/or platforms

## CORBA

- ◆ Common Object Request Broker Architecture
- ◆ Use Interface Definition Language (IDL) to describe service interface
- ◆ Provide mappings from IDL to other languages such as Java, C++, and so on.



## IDL Example

```
module bank {  
    interface BankAccount {  
        exception ACCOUNT_ERROR { long errcode; string message;};  
  
        long querybalance(in long acnum) raises (ACCOUNT_ERROR);  
        string queryname(in long acnum) raises (ACCOUNT_ERROR);  
        string queryaddress(in long acnum) raises (ACCOUNT_ERROR);  
  
        void setbalance(in long acnum, in long balance) raises (ACCOUNT_ERROR);  
        void setaddress(in long acnum, in string address) raises (ACCOUNT_ERROR);  
    };  
};
```

## Web Services

- ◆RPC over HTTP
  - Client and server communicate using HTTP requests and responses

## Web Service Example – HashService

1. Create a web application
2. Create a POJO
  - @WebService
  - @WebMethod
3. Run the application

Done!

## Metro

- ◆ <http://metro.java.net/>
- ◆ A Java web service library backed by SUN/Oracle
- ◆ Implementation of the latest Java web service specifications
- ◆ Guaranteed interoperability with .NET Windows Communication Foundation (WCF) web services
- ◆ Easy to use

## Other Java Web Service Libraries

- ◆ Apache Axis
  - <http://ws.apache.org/axis/>
- ◆ Apache Axis2
  - <http://axis.apache.org/axis2/java/core/>
- ◆ Apache CXF
  - <http://cxf.apache.org/>

## GlassFish

- ◆ <http://glassfish.java.net/>
- ◆ An Java EE application server backed by SUN/Oracle
- ◆ Complete implementation of Java EE specifications
  - i.e. not just a *servlet container* like Tomcat
  - Built-in web service support using Metro

## Java Annotations

- ◆ Available since JDK 1.5 (Java 5)
- ◆ Data about a program that is not part of the program itself
- ◆ Can be used by compiler, VM, and other software tools for *various purposes*

## Annotation Examples ...

- ◆ Error detection

```
@Override  
public String toString()
```

- ◆ Suppress warning

```
@SuppressWarnings("unchecked")  
public List<User> getAllUsers()
```

## ... Annotation Examples ...

- ◆ Servlet mapping in Servlet 3.x Specification

```
@WebServlet(name="Hello", urlPatterns="/Hello")  
public class HelloServlet extends HttpServlet
```

- ◆ Web service

```
@WebService  
public class HashService {  
  
    @WebMethod  
    public String md5( String text )  
}
```

## ... Annotation Examples

- ◆ Hibernate mapping

```
public class Employee {  
  
    @Id  
    private Integer id;  
  
    @Basic  
    private String name;  
  
    @ManyToOne  
    private Employee supervisor;  
}
```

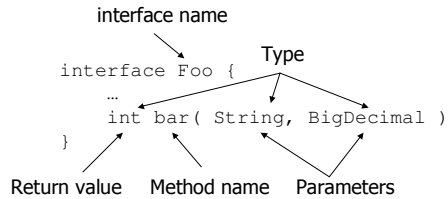
## WSDL

- ◆ A language for describing web services
  - Where the service is
  - What the service does
  - How to invoke the operations of the service
- ◆ Plays a role similar to IDF in CORBA

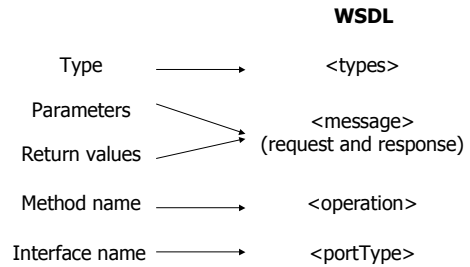
## Sample WSDL Documents

- ◆ HashService -  
<http://localhost:8084/ws/HashServiceService?wsdl>
  - /<AppName>/<ServiceName>Service?wsdl
- ◆ Amazon ECS -  
<http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl>

## How Do We Describe an API



## How Do We Describe a Web Service API



## Web Service Example – Consume HashService

- ◆ Generate client side interface and stub from WSDL
  - Using Metro's `wsimport`
  - A bit tricky due to an older JAX-WS API shipped with JDK 6
    - ◆ Copy `webservicess-api.jar` to `jre/lib/endorsed/`
- ◆ Write client code

## SOAP

- ◆ <http://www.w3.org/TR/soap/>
- ◆ Simple Object Access Protocol

## A Sample SOAP Message

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
  xmlns:xsi=http://www.w3.org/1999/XMLSchema-instance
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doSpellingSuggestion xmlns:ns1="urn:GoogleSearch"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <key xsi:type="xsd:string"> 00000000000000000000000000000000</key>
      <phrase xsi:type="xsd:string"> britney spears</phrase>
    </ns1:doSpellingSuggestion>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## SOAP Encoding

- ◆ <http://schemas.xmlsoap.org/encoding>
- ◆ Include all built-in data types of XML *Schema Part 2: Datatypes*
  - `xsi` and `xsd` name spaces

## SOAP Encoding Examples

```
int a = 10;      <a xsi:type="xsd:int">10</a>
float x = 3.14159; <x xsi:type="xsd:float">3.14159</x>
String s = "SOAP"; <s xsi:type="xsd:string">SOAP</s>
```

## Compound Values and Other Rules

```
<iArray xsi:type=SOAP-ENC:Array SOAP-ENC:arrayType="xsd:int[3]">
  <val>10</val>
  <val>20</val>
  <val>30</val>
</iArray>
<Sample>
  <iVal xsi:type="xsd:int">10</iVal>
  <sVal xsi:type="xsd:string">Ten</sVal>
</Sample>
```

- ◆ References, default values, custom types, complex types, custom serialization ...

## A Sample SOAP RPC Response

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doSpellingSuggestionResponse xmlns:ns1="urn:GoogleSearch"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:string">britney spears</return>
    </ns1:doSpellingSuggestionResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

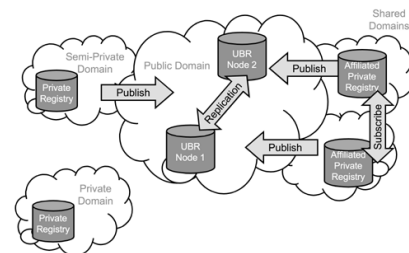
## A Sample Fault Response

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Client</faultcode>
      <faultstring>Client Error</faultstring>
      <detail>
        <m:dowJonesfaultdetails xmlns:m="DowJones">
          <message>Invalid Currency</message>
          <errorCode>1234</errorCode>
        </m:dowJonesfaultdetails>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## UDDI

- ◆ Universal Description Discovery and Integration
- ◆ A registry for web services
- ◆ A web API for publishing, retrieving, and managing information in the registry

## UDDI Registries



## Other Web Services

- ◆ Differences between web services
  - Language support
    - Single language vs. Language independent
  - Message encoding
    - Text vs. Binary
  - Transport layer
    - HTTP vs. non-HTTP
- ◆ *RESTful Web Services*

## REST

- ◆ Representational State Transfer
- ◆ Introduced by Roy Fielding in his Ph.D. dissertation on network-based software architecture

## Common Characteristics of RESTful Web Services

- ◆ Access through URL instead of method calls
- ◆ Request and response in XML or JSON
- ◆ Stateless
- ◆ Use HTTP request methods explicitly

## Map HTTP Request Methods to CRUD

- | ◆ HTTP Method | ◆ Data management |
|---------------|-------------------|
| ■ POST        | ←→ ■ Create       |
| ■ GET         | ←→ ■ Retrieve     |
| ■ PUT         | ←→ ■ Update       |
| ■ DELETE      | ←→ ■ Delete       |

## RESTful Web Service Example

- ◆ Manage student data
  - List
  - Add
  - Get
  - Update
  - Delete

## Sample Data

```
<students>
  <student>
    <name>Joe</name>
    <age>20</age>
  </student>
  <student>
    <name>Jane</name>
    <age>21</age>
  </student>
</students>
```

## HTTP Request - List All Students

```
GET /students HTTP 1.1
Host: myserver
```

## HTTP Request – Add A Students

```
POST /students/Tom HTTP 1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<student>
  <name>Tom</name>
  <age>18</age>
</student>
```

## HTTP Request – Get A Students

```
GET /students/Tom HTTP 1.1
Host: myserver
```

## HTTP Request – Update A Students

```
PUT /students/Tom HTTP 1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<student>
  <name>Tom</name>
  <age>19</age>
</student>
```

## HTTP Request – Delete A Students

```
DELETE /students/Tom HTTP 1.1
Host: myserver
```

## Advantages of RESTful Web Services

- ◆ Do not depend on complex specifications and library, i.e. easy to create
- ◆ Language independent, i.e. easy to use
- ◆ Take full advantage of infrastructure support for HTTP, e.g. caching



## Summary

- ◆ RPC and RMI
- ◆ CORBA
  - IDL
- ◆ SOAP, WSDL, UDDI
  - Create and consume SOAP web services
- ◆ RESTful web services