Client-Server Architecture

Client-Server Example

Client-Server Interaction as Function Calls

Socket Programming – Client

RPC and RMI
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
1. Lookup
2. Stub (proxy)
3. Method invocation
4. Parameters
5. Result
6. Return result

Cross Platform RPC
- The client and the server use different languages and/or platforms
- C/C++, Java, C#, Python, PHP...

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**

```java
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

- `@WebService`
- `@WebMethod`
- `web.xml`
- `sun-jaxws.xml`
  - `<endpoint>`

**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 Axis2**
  - `http://axis.apache.org/axis2/java/core/
- **Apache CXF**
  - `http://cxf.apache.org/

**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

- Amazon ECS - http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl

How Do We Describe an API

```java
interface Foo {
    int bar(String, BigDecimal);
}
```

Return value   Method name   Parameters

How Do We Describe a Web Service API

- **Interface Name**: WSDL
- **Type**: <types>
- **Parameters**: <message>
- **Return values**: (request and response)
- **Method name**: <operation>
- **Interface name**: <portType>

Web Service Example: Consume HashService

- Generate client side interface and stub from WSDL using Metro's `wsimport`
- Write client code

A Sample SOAP Message

```xml
<?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"/>

    <ns1:key xsi:type="xsd:string">00000000000000000000000000000000</ns1:key>
    <ns1:phrase xsi:type="xsd:string">britney spears</ns1:phrase>
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

SOAP

- http://www.w3.org/TR/soap/
- Simple Object Access Protocol
SOAP Encoding

- http://schemas.xmlsoap.org/encoding
- Include all built-in data types of XML
  - xsi and xsd name spaces

SOAP Encoding Examples

```xml
int a = 10;  
<ns xsi:type="xsd:int">10</ns>

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

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

.Sample>
  <Val xsi:type="xsd:int">10</Val>
  <Val xsi:type="xsd:string">Ten</Val>
</Sample>
```

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

A Sample SOAP RPC Response

```xml
<?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

```xml
<?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>
    <SOAP-ENV:Fault>  
      <faultcode>SOAP-ENV:Client</faultcode>  
      <faultstring>Client Error</faultstring>  
      <m:dowJonesFaultDetails xmlns:m="DowJones">  
        <message>Invalid Currency</message>  
        <errorcode>1234</errorcode>  
      </m:dowJonesFaultDetails>  
      <m:dowJonesFaultDetails>  
        <message>Invalid Currency</message>  
        <errorcode>1234</errorcode>  
      </m:dowJonesFaultDetails>  
    </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-base software architecture

The REST Constraints

- Client and server
- Stateless
- Support caching
- Uniformly accessible
- Layered
- (Optional) support code-on-demand

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

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Data management</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>Create</td>
</tr>
<tr>
<td>GET</td>
<td>Retrieve</td>
</tr>
<tr>
<td>PUT</td>
<td>Update</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete</td>
</tr>
</tbody>
</table>
RESTful Web Service Example

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

Sample Data

```xml
<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 using Metro
- RESTful web services

Further Readings

- [Java Web Services Up and Running](#) by Martin Kalin
- [RESTful Java Web Services](#) by Jose Sandoval
- [The Rise and Fall of CORBA](#) by Michi Henning