Client-Server Architecture

Client-Server Example

Client-Server Interaction as Function Calls

Remote Procedure Call

Socket Programming – Client

Client side:

User user = auth( username, password );

Server side:

User auth( String username, String password )
{
  ...
  if ( isValid ) return user;
  else return null;
}

Remote Procedure Call (RPC)
  - C
  - CORBA
    - Cross platform
    - Interface Definition Language (IDL)
  - Remote Method Invocation (RMI)
    - Java
  - Web services
    - XML as IDL
RMI

```
Client code
1. lookup
2. Stub (proxy)
3. Method invocation
4. Return value
Type of the local object??
```

Interface in RMI

- Must extends `java.rmi.Remote`
- Shared by both client and server code
- E.g. `AuthInterface`

```java
class AuthInterface extends java.rmi.Remote {
    User auth(String username, String password) throws java.rmi.RemoteException;
```

More About RMI

- Compilation and Execution

Spring RMI Support

- POJO interface and implementation
- `RmiServiceExporter` handles remote object registration, lookup, stub generation etc.
- `service`, `serviceInterface`, and `serviceName`
- `registryPort`
- Evelyn example

Alternatives to RMI

<table>
<thead>
<tr>
<th>Name</th>
<th>Language</th>
<th>Message Type</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMI</td>
<td>Java-to-Java</td>
<td>Binary</td>
<td>Default 1099</td>
</tr>
<tr>
<td>Hessian</td>
<td>Mostly Java-to-Java</td>
<td>Binary</td>
<td>HTTP</td>
</tr>
<tr>
<td>Burlap</td>
<td>Any</td>
<td>XML</td>
<td>HTTP</td>
</tr>
<tr>
<td>Spring HTTP Invoker</td>
<td>Java-to-Java</td>
<td>Binary</td>
<td>HTTP</td>
</tr>
<tr>
<td>Web services</td>
<td>Any</td>
<td>XML</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

Web Services

- Roughly speaking, anything that encodes RPC calls in `XML messages` and transport them over `HTTP`
- Simple Object Access Protocol (SOAP)
- Web Service Description Language (WSDL)
- Universal Description, Discovery, and Integration (UDDI)
SOAP

- http://www.w3.org/TR/soap/
- Format conventions for message content and routing directions in the form of an *envelope*
- Rules for encoding custom data types
- Application of the envelop and the data encoding rules for representing RPC calls and responses
- Transport protocol binding (usually HTTP)

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">
      <key xsi:type="xsd:string">00000000000000000000000000000000</key>
      <phrase xsi:type="xsd:string">britney speers</phrase>
    </ns1:doSpellingSuggestion>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Things to Note

- Namespaces
- `<Envelope>`
  - Optional `<Header>` - for information related to processing of the message
  - `<Body>`
- `encodingStyle`
- `<Fault>`
  - Only sub-element of `<Body>` defined by SOAP

SOAP Encoding

```xml
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

```xml
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

- References, default values, custom types, root attribute, complex types, custom serialization ...
SOAP RPC Elements
◆ Target object URI in HTTP header
◆ Namespace qualified method name and method parameters
◆ Optional SOAP header for additional data that’s not part of the parameter list

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">
<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>

Apache Axis
◆ http://ws.apache.org/axis/
An implementation of SOAP
◆ Simplifies producing and consuming web services
  ◆ Create WSDL document from Java source code
  ◆ Create Java classes from WSDL document
  ◆ Encode and decode XML requests and responses
  ◆ ...

Provide SOAP Services with Axis and Spring
◆ Axis configuration
  ◆ Axis servlet in web.xml
  ◆ server-config.wsdd under /WEB-INF
◆ Spring-related code
  ◆ JaxRpc wrapper around POJO service object

Access SOAP Services with Spring and Axis
◆ Service bean configuration
  ◆ serviceFactoryClass
  ◆ wsdlDocumentUrl
  ◆ namespaceUri
  ◆ serviceName
  ◆ portName
  ◆ serviceInterface
WSDL
- A language for describing web services
  - What the service does
  - Where is the service
  - How to invoke the operations of the
    service
- Why do we need WSDL when we have API documentation??

Sample WSDL Documents
- Amazon ECS -
  http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl
- Google Web APIs -
  http://api.google.com/GoogleSearch.wsdl

How Do We Describe an API?

How Do We Describe an Web Service API?

A Little More Details
- The name attribute uniquely identifies a
  <message>, an <operation>, or a
  <portType>
- Operation behavior patterns
  - Request-response
  - Solicit-response
  - One-way
  - Notification
- <fault>

Other WSDL Elements
- <definitions>
  - targetNamespace
- <import>
- <binding> - concrete protocol and format
  specification for a <portType>
  - E.g. <input> should be in SOAP header or body,
    what encoding rules should be used etc.
- <service>

WSDL

<types>
<message>
<operation>
<portType>
Service

- `<service>`: One or more ports
- `<port>`: Where the operations can be accessed (endpoint)
- `<binding>`: Implementation of the operations with a certain protocol
- `<portType>`: A set of operations

JAX-RPC

- A specification for building XML-based web services and clients using Java

A Few JAX-RPC Concepts

- **Stub (proxy) and Tie (skeleton)**
- **Holder**
  - IN, OUT, and INOUT parameters

Service Invocation Patterns

- **Static binding**
  - statically generated stub
- **Dynamic binding**
  - statically generated interface
  - `javax.xml.rpc.Service.getPort()`
- **Dynamic Invocation Interface (DII)**
  - `javax.xml.rpc.Call`

UDDI

- A registry for web services
  - Information about the service providers
  - Classifications of services
  - Technical information about the service interfaces
- A web API for publishing, retrieving, and managing information in the registry

Registries
Core Data Types

- `<businessEntity>`
- `<businessService>`
- `<bindingTemplate>`
- `<tModel>`

**<businessEntity>**

- Information about the service provider

```xml
<businessEntity businessKey="uuid:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx">
  <name>Some Company</name>
  <description>We provide web services</description>
  <contacts>...</contacts>
  <businessServices>...</businessServices>
</businessEntity>
```

**<businessService>**

- Descriptive information about services

```xml
<businessService serviceKey="uuid:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx">
  <name>Hello World</name>
  <description>A great web service</description>
  <bindingTemplates>...</bindingTemplates>
</businessService>
```

**<bindingTemplate>**

- Technical information about services

```xml
<bindingTemplate bindingKey="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx">
  <description xml:lang="en">SOAP binding for Hello World</description>
  <accessPoint URLType="http">http://localhost:8080/soap</accessPoint>
  <tModelInstanceDetails>
    <tModelInstanceInfo tModelKey="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx" />
  </tModelInstanceDetails>
</bindingTemplate>
```

**<tModel>**

- Interface specification about services

```xml
<tModel TModelKey="uuid:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx">
  <name>Hello World Port Type</name>
  <description>Interface for a great web service</description>
  <overviewDoc>
    <overviewURL>http://localhost:8080/soap/helloworld.wsdl</overviewURL>
  </overviewDoc>
</tModel>
```

**UDDI APIs**

- **Node API Sets**
  - Interaction among registry nodes
- **Client API Sets**
  - Publish services to a registry
  - Search a registry for services
WSDL for UDDI Client API

- http://www.uddi.org/wsdl/publish_v2.wsdl
- http://www.uddi.org/wsdl/inquire_v2.wsdl