Managing Data on the World Wide-Web

Web Services

Elad Kravi

Based on S. Bussel presentation
Overview

1. Introduction, RPC/RMI
2. SOAP, WSDL, UDDI
3. AXIS
Web Service

• A software system designed to support interoperable machine-to-machine interaction over a network
  – **Interoperability**: The capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units

• Self-contained, modular Web application that can be published, located and invoked across the Web

• A web service can perform functions of varying complexities

• With web services, a system that runs on a Win 2k server can connect with a UNIX server
Why is it Difficult to Use Ordinary Web Sites as Services?

• Consider an application that should return the price of the book “The Hobbit”
  – the application needs to fill the required form and launch a “browser-like” request
  – the application needs to parse the page, based on its specific HTML structure, and retrieve the price
  – once the Web site changes its user interface, your program is broken
  – how can we find new online stores?
• It would help if we could call functions, such as:
  – Amazon.getPrice("The Hobbit")
  – Amazon.buyBook("The Hobbit", myId)
• **Reusable application-components**
  – There are things applications need very often. So why make these over and over again?
  – Web services can offer application-components like: currency conversion, weather reports, or even language translation as services

• **Connect existing software**
  – Web services can help to solve the interoperability problem by giving different applications a way to link their data
  – With Web services you can exchange data between different applications and different platforms
Difficulties in Using Remote Functions

• **Different coding languages:**
  For each remote function, we need to phrase a call in the specific coding language that is used for its implementation

• **Knowing the method signature:**
  For each remote function, we need to contact the provider in order to find out what exactly the signature is
  – i.e., parameters, return value, what are the type

• **Listening to the right socket:**
  On the server side, each remote function needs to listen to a server socket bound to a specific port
  – not in line with protection by firewalls
• **Data transfer:**
  – use an agreed interface and a syntax that all applications are familiar with, e.g., XML
    • For example, **SOAP**
  – use HTTP to transfer data through port 80
  – another approach: **REST** (next week)

• **Understanding the signature:**
  – use a standard for publishing methods, their signatures and their usage
    • For example, **WSDL**
  – by obtaining (only) the WSDL the client can call the remote service.

• **Locating services:**
  – use standard directory structures for publishing available services
    • For example, **UDDI**
Remote Procedure Calls (RPC)

- A remote procedure call makes a call to a remote service
  - Looks like a local call

- RPCs are naturally accommodated in the client-server model
  - An RPC is initiated by the client, which sends a request message to a known remote server to execute a specified procedure with supplied parameters
RPC – What has changed?

- RPC allows applications to become distributed transparently.
- RPC makes architecture of remote machine transparent;
  - Java can invoke methods that were not written in Java (e.g., .NET methods) and vice versa.
- Callers must deal with network failures without knowing whether the remote procedure was actually invoked;
  - Easy to handle: idempotent procedures (i.e., procedures that have no additional effects if called more than once).
When the software in question uses object-oriented principles, RPC might be called remote method invocation (RMI).

The **Java RMI** API allows an object running in one Java virtual machine to invoke methods on an object running in another Java virtual machine.

RMI applications often comprise two separate programs, a server and a client.

- Such an application is sometimes referred to as a distributed object application.
RMI – Remote Method Invocation

- A typical server program creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects.

- A typical client program obtains a remote reference to one or more remote objects on a server and then invokes methods on them.
To obtain references to remote objects, there exists a registry, called the RMI registry, which is a remote object that maps names to remote objects.
public interface RmiServerIntf extends Remote {
    public String getMessage() throws RemoteException;
}

public class RmiServer
    extends UnicastRemoteObject implements RmiServerIntf {
    public static final String MESSAGE = "Hello World";

    public String getMessage() {
        return MESSAGE;
    }

    public static void main(String args[]) throws Exception {
        ...
        // Instantiate RmiServer
        RmiServer skeleton = new RmiServer();

        // Bind this object instance to the name "RmiServer"
        Naming.rebind("//localhost/RmiServer", skeleton);
    }
    ...
}
import java.rmi.Naming;

public class RmiClient {
    public static void main(String args[]) throws Exception {
        RmiServerIntf stub =
            (RmiServerIntf) Naming.lookup("//localhost/RmiServer");
        System.out.println(stub.getMessage());
    }
}
Stubs and Skeletons

- RMI uses a standard mechanism (employed in RPC systems) for communicating with remote objects: *stubs* and *skeletons*

- A stub for a remote object acts as a client's local representative or proxy for the remote object

- The caller invokes a method on the local stub which is responsible for carrying out the method call on the remote object
The Stub

• When a stub's method is invoked, it does the following:
  i. initiates a connection with the remote JVM containing the remote object
  ii. marshals (writes and transmits) the parameters to the remote JVM
  • marshaling is similar to serialization and refers to the process of converting objects into a byte-stream in order to transmit the data to remote objects
  iii. waits for the result of the method invocation
  iv. unmarshals (reads) the return value or exception returned
  v. returns the value to the caller
The Skeleton

- In the remote JVM, each remote object has a corresponding skeleton.
- The skeleton is responsible for dispatching the call to the actual remote object implementation.
- When a skeleton receives an incoming method invocation, it does the following:
  1. Unmarshals (reads) the parameters for the remote method.
  2. Invokes the method on the actual remote object implementation.
  3. Marshals (writes and transmits) the result (return value or exception) to the caller.
The mechanism of stubs and skeletons hides the serialization of parameters and the network-level communication in order to present a simple invocation mechanism.
Service-Oriented Architecture (SOA):

- SOA is an architecture approach for defining, linking, and integrating reusable services that have clear boundaries and are self-contained with their own functionalities.
- Within this type of architecture, we can orchestrate the services in business-processes/applications.
- SOA provides interoperability, reusability and extensibility, and suited for heterogeneous environments.
- Web services support service-oriented architectures, using in particular XML and HTTP.
Overview

1. Introduction, RPC/RMI
2. SOAP, WSDL, UDDI
3. AXIS
What is SOAP?

• A protocol for invoking web Services
  – XML based
  – applications may exchange information over HTTP
    • HTTP is just one possible communication pattern supported by SOAP (e.g. SMTP. See specs and an example)

• A SOAP message is an ordinary XML document containing the following elements:
  – Envelope: identifies the XML document as a SOAP message: required
  – Header – contains header information: optional
  – Body – contains call or response information: required
  – Fault – provides information about errors that occurred while processing the message: optional
Example – SOAP Request

POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 312

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
   <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
   </m:GetStockPrice>
</soap:Body>

</soap:Envelope>

In this example, a GetStockPrice request is sent to a server.

Required for SOAP messages

application-specific elements
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 321

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
        <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>

In this example, a GetStockPrice request is sent to a server.
Web Services Description Language (WSDL)

- WSDL is used to describe web services
- Written in XML
- A **WSDL document** describes how to access a web service and what operations it will perform
- It specifies the location of the service, and the methods of the service, using these major elements:
  - `<types>`: Defines the data types used by the web service
  - `<message>`: Defines the data elements for each operation
  - `<portType>`: Describes the operations that can be performed and the messages involved
  - `<binding>`: Defines the protocol and data format for each port type
WSDL - Example

```xml
<message name="getTermRequest">
    <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
    <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
    <operation name="getTerm">
        <input message="getTermRequest"/>
        <output message="getTermResponse"/>
    </operation>
</portType>
```

<table>
<thead>
<tr>
<th>Name of a port</th>
<th>Name of an operation</th>
<th>Input message</th>
<th>Output message</th>
</tr>
</thead>
<tbody>
<tr>
<td>glossaryTerms</td>
<td>getTerm</td>
<td>getTermRequest</td>
<td>getTermResponse</td>
</tr>
</tbody>
</table>
Demo - GeoIPService

- GeoIPService enables to look up countries by IP address / Context

- WSDL:
  [http://www.webservicex.net/geoipservice.asmx?WSDL](http://www.webservicex.net/geoipservice.asmx?WSDL)

- Sending SOAP messages online:
  - [http://www.webservicex.net/geoipservice.asmx?op=GetGeoIP](http://www.webservicex.net/geoipservice.asmx?op=GetGeoIP)
• **Universal Description, Discovery, and Integration**

• UDDI is a mechanism for *registering* and *discovering* Web services
  – Think of UDDI as a telephone book

• for the applications we have considered so far, an external web service would typically be discovered manually by the programmer and be hardwired into the application. This is known as **static discovery**
• Apache jUDDI (pronounced "Judy") is an open source implementation of OASIS's UDDI v3 specification

• UDDI communicates via SOAP

• A UDDI business registration consists of three components:
  – White Pages — address, contact, and known identifiers
  – Yellow Pages — industrial categorizations based on standard taxonomies
  – Green Pages — technical information about services exposed by the business
• Example of how we can locate web services using UDDI: https://msdn.microsoft.com/en-us/library/ms950813.aspx

• UDDI has yet seen widespread adoption
Overview

1. Introduction, RPC/RMI
2. SOAP, WSDL, UDDI
3. AXIS
The Apache Axis2 project is a Java-based implementation of both the client and server sides of the web services equation.

Axis is essentially a SOAP engine – a framework for constructing SOAP processors:
- send, receive and process SOAP messages
- create a Web service out of a plain Java class
- create implementation classes for both the server and client using WSDL
- easily retrieve the WSDL for a service
- create or utilize a REST-based Web service
- And more…

Differences from Axis1 - http://axis.apache.org/axis2/java/core/docs/migration.html
What We Would Like to Create

- **Client applications**: applications that can call a remote Web service
- **Services**: methods that can be called by remote applications
- **Service descriptions**: WSDL files that describe our methods
• We will build a web service (both server and client side) using Eclipse WTP (already integrated in your J2EE Eclipse version)

• The demo is based on the short YouTube tutorial Create web service with Eclipse + Axis2 + Tomcat
Deploying The Service

- One option, in which you include the entire Axis2 application is:
  - Export the completed project to WAR
  - Put this WAR under Tomcat webapps directory
  - That’s it!

- Usually this is not required, and all you need to deploy is the service only, while Axis2 is already deployed.

- For this you will need to put axis2.war file under $CATALINA_BASE/webapp/
  - Download Axis2 version 1.6.3 WAR distribution (use this link)
References

- https://docs.oracle.com/javase/tutorial/rmi/
- SOA fundamentals in a nutshell
- http://www.w3schools.com/xml/xml_services.asp
- http://www.w3.org/TR/wsl
- YouTube tutorial: Create web service with Eclipse + Axis2 + Tomcat