Multitiered Architecture
Web Services

Benoît Garbinato
Outline

- Definition and origin of web services
- General web services
- RESTful web services
- Using JAX-WS (General web services)
- Using JAX-RS (RESTful web services)
Web Service – Definitions

A service offered by an electronic device to another electronic device, communicating with each other via the world wide web.

\textit{rmi* on http}

*\textit{rmi} $\Rightarrow$ remote method invocation

Software system designed to support interoperable machine-to-machine interaction over a network.

World wide web consortium (w3c)
RMI | Fundamental Idea

A remote method (procedure) is transparently invoked (called) across the network, as if it was local.
RMI | Some history

1979  Bill Joy introduces the “Berkeley Enhancements”, mainly interprocess communication (IPC) facilities. The modern network Unix is born (BSD).

mid 80’s  Sun Microsystems uses BSD Unix as operating system for their workstations. They extends it with RPC, on top of which they build NFS and NIS (later on NIS+).

1988  The Open Software Foundation (OSF) is formed to develop a portable open system platform, known as the Distributed Computing Environment (DCE). The latter proposes DCE RPC as basic communication mechanism.

mid 90’s  The Object Management Group (OMG) follows the same approach to devise the Common Object Request Broker Architecture (CORBA) for object-based middleware. At the same time, Sun greatly simplifies & extends the RMI paradigm its Java & Jini platforms.

Today  Web Services are a widespread approach to invoke remote services on the web but they are really just a web-flavored version of the good old RPC/RMI paradigm, using HTTP & XML/JSON.
A glimpse at the CORBA promise

Common Object Request Broker Architecture

Interface Definition Language (IDL)

Stub

Skeleton

Object Request Broker (ORB)
A glimpse at the CORBA reality

**Common Object Request Broker**
Local Method Invocations

```
result = server.foo(parameters)
```

Diagram showing a call stack with parameters and a result, linking a client to a server.
Remote Method Invocation

- How is the remote service specified?
- How is data in the request/response represented?
- What is the underlying transport protocol?
The OSI model | Reminder

Client:
- Application
- Presentation
- Session
- Transport
- Network
- Data link
- Physical link

Server:
- Application
- Presentation
- Session
- Transport
- Network
- Data link
- Physical link

Questions:
- How is the remote service specified?
- How is data in the request/response represented?
- What is the underlying transport protocol?
The OSI model | Java RMI

- **client**
  - application
  - presentation
  - session
  - transport
  - network
  - data link

- **server**
  - application
  - presentation
  - session
  - transport
  - network
  - data link

**Java interfaces**

- **Java serialization**
  - JRMP* or IIOP†

*Java Remote Method Protocol
†Internet Inter-ORB Protocol
The OSI model | CORBA

- **client**
  - application
  - presentation
  - session
  - transport
  - network
  - data link

- **server**
  - application
  - presentation
  - session
  - transport
  - network
  - data link

**Interface Definition Language (IDL)**
- language-specific mappings
- Internet Inter-ORB Protocol (IIOP)

**Physical link**
Service specification with Java RMI

```java
public interface CalendarService extends Remote {
    public DayCalendar createCalendar(String name) throws RemoteException, CalendarException;
    public DayCalendar getCalendar(String name) throws RemoteException, CalendarException;
    public ArrayList getCalendars() throws RemoteException;
    public boolean exists(String name) throws RemoteException;
}
```

```java
public class CalendarServer extends UnicastRemoteObject implements CalendarService {
    ...
    public DayCalendar createCalendar(String name) throws RemoteException, CalendarException {
        if (calendars.containsKey(name)) throw new CalendarException(name + " already exists.");
        DayCalendar newCal = new DayCalendarImpl(name);
        calendars.put(name, newCal);
        return newCal;
    }
    public DayCalendar getCalendar(String name) throws RemoteException, CalendarException {
        if (!calendars.containsKey(name)) throw new CalendarException(name + " does not exist.");
        return ((DayCalendar) calendars.get(name));
    }
    public ArrayList getCalendars() throws RemoteException {
        return new ArrayList(calendars.values());
    }
    public boolean exists(String name) throws RemoteException {
        return calendars.containsKey(name);
    }
    ...
}
```
module employee {
    struct EmployeeInfo {
        wstring name;
        long number;
        double salary;
    };
}

exception SQLError {
    wstring message;
};

interface Employee {
    EmployeeInfo getEmployee (in wstring name) raises (SQLError);
    EmployeeInfo getEmployeeForUpdate (in wstring name) raises (SQLError);
    void updateEmployee (in EmployeeInfo name) raises (SQLError);
};
The OSI model | Web Services

<table>
<thead>
<tr>
<th>client</th>
<th>server</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>application</td>
</tr>
<tr>
<td>presentation</td>
<td>presentation</td>
</tr>
<tr>
<td>session</td>
<td>session</td>
</tr>
<tr>
<td>transport</td>
<td>transport</td>
</tr>
<tr>
<td>network</td>
<td>network</td>
</tr>
<tr>
<td>data link</td>
<td>data link</td>
</tr>
<tr>
<td>physical link</td>
<td>Physical link</td>
</tr>
</tbody>
</table>

The OSI model consists of seven layers, each serving a specific function. The client and server layers interact through various protocols and specifications.

- **Network Layer**: This layer is responsible for routing data across the network. Example: HTTP (Hypertext Transfer Protocol).
- **Transport Layer**: This layer ensures reliable delivery of data. Example: TCP (Transmission Control Protocol).
- **Session Layer**: This layer manages the session and connection between the client and server. Example: SSL (Secure Sockets Layer).
- **Presentation Layer**: This layer deals with the representation of data. Example: JSON (JavaScript Object Notation) / XML (eXtensible Markup Language).
- **Application Layer**: This layer provides specific services for the client application. Example: WSDL (Web Services Description Language) / WADL (Web Application Description Language).
- **Data Link Layer**: This layer provides the physical connection between the network and the transport layer. Example: Ethernet.
- **Physical Layer**: This layer deals with the transmission of raw bits over a physical medium. Example: Ethernet cables.

**How is data in the request/response represented?**

- JSON / XML / ...

**How is the remote service specified?**

- WSDL / WADL

**What is the underlying transport protocol?**

- HTTP
Web service types

There exists two types of web services:

- **General web services**, which provide support to remotely call any kind of operations
- **RESTful web services**, which focus on state transfer

REST = Representational State Transfer

They both require a significant amount of boilerplate code to function

General web services

- They provide support to remotely call any kind of operations
- They rely on the Web Services Description Language (WSDL)
- They rely on the Simple Object Access Protocol (SOAP), an XML standard defining a message architecture and format
- In Java, JAX-WS is the technology that encapsulates (part of) the complexity of defining and using general web services
Using JAX-WS (1)

WSDL

Multitiered Architecture | Web Services © Benoit Garbinato
hello Method invocation

Method parameter(s)

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>Tim</td>
</tr>
</tbody>
</table>

Method returned
java.lang.String : "Hello Tim !"

SOAP Request
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:hello xmlns:ns2="http://service/">
            <name>Tim</name>
        </ns2:hello>
    </S:Body>
</S:Envelope>

SOAP Response
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:helloResponse xmlns:ns2="http://service/">
            <return>Hello Tim !</return>
        </ns2:helloResponse>
    </S:Body>
</S:Envelope>
Using JAX-WS (3)

```java
@WebService(serviceName = "HelloWorld")
@Stateless()
public class HelloWorld {
    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String name) {
        System.out.println("I received a call with " + name + " as parameter");
        return "Hello " + name + " !";
    }
}
```

```java
public class Client {
    public static void main(String[] args) {
        String response = hello("Tim");
        System.out.println("response = " + response);
    }

    private static String hello(java.lang.String name) {
        webservice.HelloWorld_Service service = new webservice.HelloWorld_Service();
        webservice.HelloWorld port = service.getHelloWorldPort();
        return port.hello(name);
    }
}
```
Using JAX-WS (4)
RESTful web services (1)

- They focus on state transfer, usually from/to some persistent storage, e.g., a relational database.
- They manipulate state as resources accessed using Uniform Resource Identifiers (URIs) and four HTTP verbs as CRUD operations.

<table>
<thead>
<tr>
<th>Uniform Resource Identifier</th>
<th>GET</th>
<th>PUT</th>
<th>POST</th>
<th>DELETE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection, such as:</td>
<td>list</td>
<td>replace the entire collection with another collection</td>
<td>create a new item in the collection, and return its automatically assigned URI</td>
<td>delete the entire collection</td>
</tr>
<tr>
<td><a href="http://myservice.com/items/">http://myservice.com/items/</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Item, such as:</td>
<td>retrieve</td>
<td>replace the addressed item of the collection</td>
<td>not often used. Treat the addressed item as a collection and create a new item in it</td>
<td>delete the addressed item</td>
</tr>
<tr>
<td><a href="http://myservice.com/items/item17">http://myservice.com/items/item17</a></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HTTP</th>
<th>CRUD</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUT</td>
<td>CREATE</td>
</tr>
<tr>
<td>GET</td>
<td>READ</td>
</tr>
<tr>
<td>POST</td>
<td>UPDATE</td>
</tr>
<tr>
<td>DELETE</td>
<td>DELETE</td>
</tr>
</tbody>
</table>
RESTful web services (2)

- They do not formally require data to be represented using a particular format, although JSON and XML are very often used for structured data today.

- They do not formally require an interface definition language, although it is now common practice to use the Web Application Definition Language (WADL) in Java.

- In Java, JAX-RS is the technology that encapsulates (part of) the complexity of defining and using RESTful web services and Jersey is the reference implementation of JAX-RS.
Using JAX-RS (1)
```java
@Entity
@Table(name = "CUSTOMER")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Customer.findAll",
        query = "SELECT c FROM Customer c"), ...
})
public class Customer implements Serializable {
    @Id
    @Basic(optional = false)
    @NotNull
    @Column(name = "CUSTOMER_ID")
    private Integer customerId;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 10)
    @Column(name = "ZIP")
    private String zip;
    ...
    public Integer getCustomerId() {
        return customerId;
    }
    public void setCustomerId(Integer customerId) {
        this.customerId = customerId;
    }
    public String getZip() {
        return zip;
    }
    public void setZip(String zip) {
        this.zip = zip;
    }
    ...
}

@Stateless
@Path("entities.customer")
public class CustomerFacadeREST extends AbstractFacade<Customer> {
    @PersistenceContext(unitName = "CustomerDBPU")
    private EntityManager em;
    public CustomerFacadeREST() {
        super(Customer.class);
    }
    @POST
    @Override
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public void create(Customer entity) {
        super.create(entity);
    }
    @PUT
    @Path("{id}")
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public void edit(@PathParam("id") Integer id, Customer entity) {
        super.edit(entity);
    }
    @DELETE
    @Path("{id}")
    public void remove(@PathParam("id") Integer id) {
        super.remove(super.find(id));
    }
    @GET
    @Path("{id}")
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public Customer find(@PathParam("id") Integer id) {
        return super.find(id);
    }
    @GET
    @Override
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public List<Customer> findAll() {
        return super.findAll();
    }
    ...
}
```
public class CustomerDBClient {
    private WebTarget webTarget;
    private Client client;
    private static final String BASE_URI = "http://localhost:8080/CustomerDB/webresources";

    public CustomerDBClient() {
        client = javax.ws.rs.client.ClientBuilder.newClient();
        webTarget = client.target(BASE_URI).path("entities.customer");
    }

    public void create_XML(Object requestEntity) throws ClientErrorException {
        webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_XML)
            .post(javax.ws.rs.client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_XML));
    }

    public void create_JSON(Object requestEntity) throws ClientErrorException {
        webTarget.request(javax.ws.rs.core.MediaType.APPLICATION_JSON)
            .post(javax.ws.rs.client.Entity.entity(requestEntity, javax.ws.rs.core.MediaType.APPLICATION_JSON));
    }

    public String countREST() throws ClientErrorException {
        WebTarget resource = webTarget;
        resource = resource.path("count");
        return resource.request(javax.ws.rs.core.MediaType.TEXT_PLAIN).get(String.class);
    }

    public void remove(String id) throws ClientErrorException {
        webTarget.path(java.text.MessageFormat.format("{0}", new Object[]{id})).request().delete();
    }

    public void close() {
        client.close();
    }
}

public class Main {
    public static void main(String[] args) {
        CustomerDBClient client = new CustomerDBClient();

        String response = client.countREST();
        System.out.println("Client response is : ", response);
    }
    client.close();
}