Java Web Services

Intelligent Infrastructure Design for the Internet of Things

Antonio Navarro

Java Web Services

- References
- Introduction
- Publishing and invoking web services
  - Web Service Broker
  - Business Delegate
- JAX-WS
  - Introduction
  - Service Annotations
  - Customer notes

Java Web Services

- JAX-RS
  - Introduction
  - Service Annotations
  - API client
  - Example
- Conclusions

References

- Apache CXF User's Guide, http://cxf.apache.org/docs/index.html
- Bill Burke, *RESTful Java with JAX-RS 2.0, Second Edition, O'Reilly, 2013*
- B.V. Kumar, P. Narayan, T. Ng, *Implementing SOA using Java EE*, Addison-Wesley Professional, 2010
- Javatips.net, Apache CXF with Jackson, https://www.javatips.net/blog/apache-cxf-with-jackson
- JAX-RS: Data Bindings http://cxf.apache.org/docs/jax-rs-data-bindings.html#JAX-RSDataBindings-JSONsupport
- Mark D. Hansen, *SOA Using Java Web Services*, Prentice Hall, 2007
- Oracle Java Platform, Enterprise Edition (Java EE) 7, https://docs.oracle.com/javaee/7/index.html

Introduction

- Java provides two frameworks to facilitate the implementation of web services:
  - JAX-WS (Java API for XML Web Services) for SOAP
  - JAX-RS (Java API for RESTful Web Services) for REST
- Development environments make it easy to create web services using Java frameworks
- Frameworks need implementations, such as:
  - Apache CXF (JAX-WS and JAX-RS)
  - Oracle Metro (JAX-WS)
  - Oracle Jersey (JAX-RS)
- Let's take a look at the main elements of these frames and some simple examples

Publishing and invoking… Web Service Broker

- We are going to see a pattern that will help us to expose web services: the Web Service Broker.
- Purpose
  - You want to provide access to one or more services using XML and web protocols
- Also known as
  - Web Services Agent

Publishing and invoking… Web Service Broker

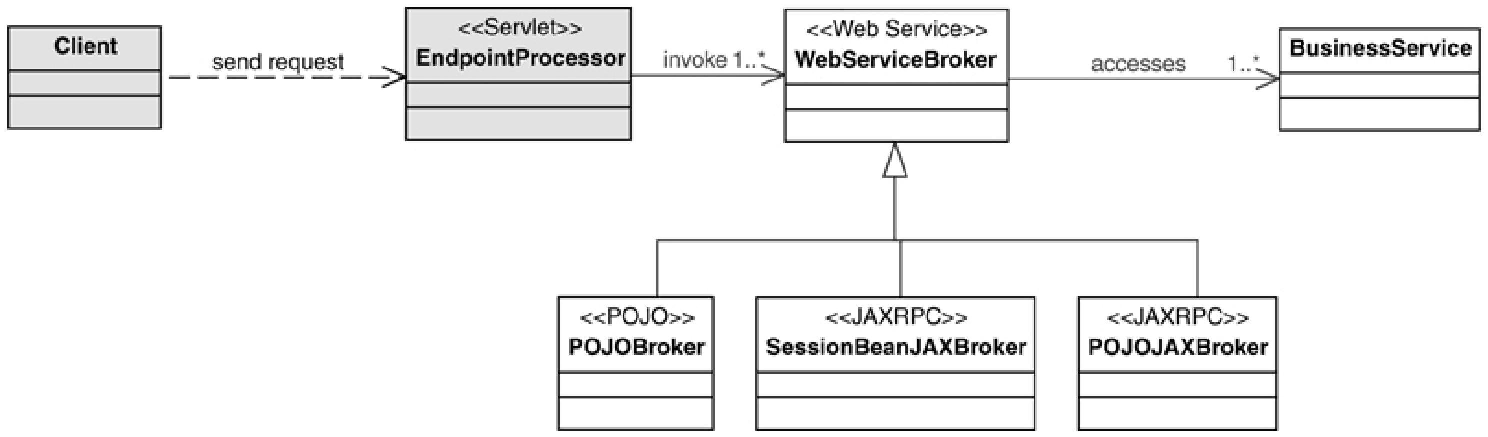
- Motivation
  - Enterprise applications expose their coarse-grained services through service facades, or application services
  - However, these services may not be ready to be exposed outside the application.
  - In addition, you may need to compose multiple services to expose them as web services:
    - Coordinating interactions between one or more services
    - Adding responses
    - Demarcating and/or clearing transactions

Publishing and invoking… Web Service Broker

- Context
  - You want to reuse and expose different services to your customers
  - You want to monitor and potentially limit the use of exposed services, based on business requirements and system resource usage.
  - Services should be exposed using open standards to enable integration of heterogeneous applications.
  - You want to bridge the gap between business requirements and existing service capabilities.
- Solution
  - Use a web service broker to expose and negotiate one or more services using XML and web protocols

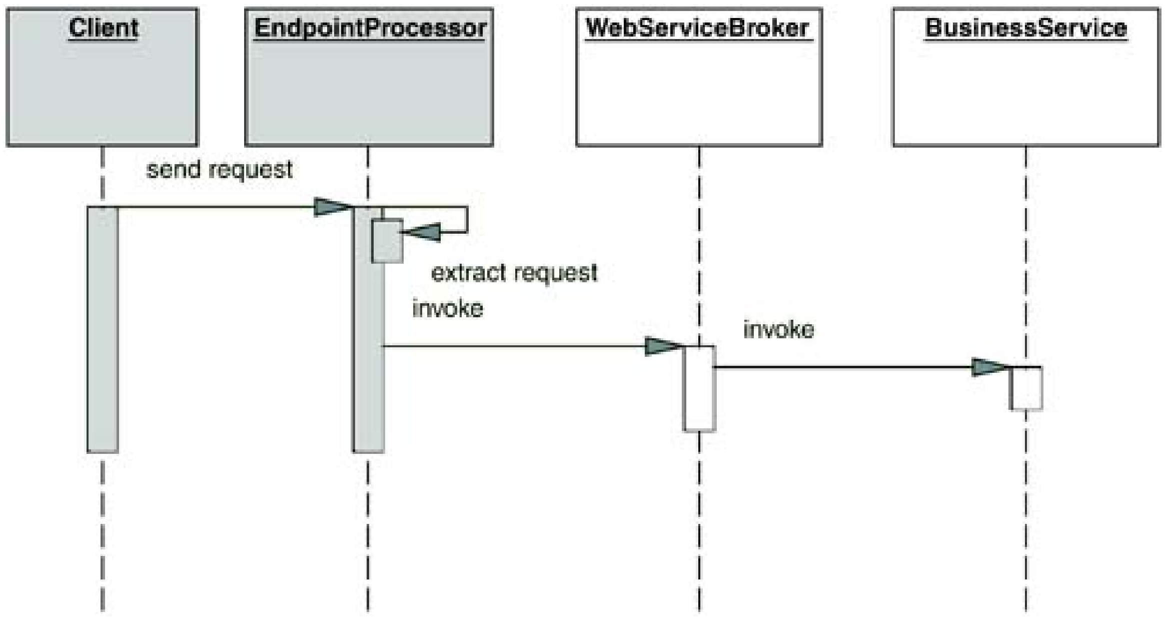
Publishing and invoking… Web Service Broker

- Description



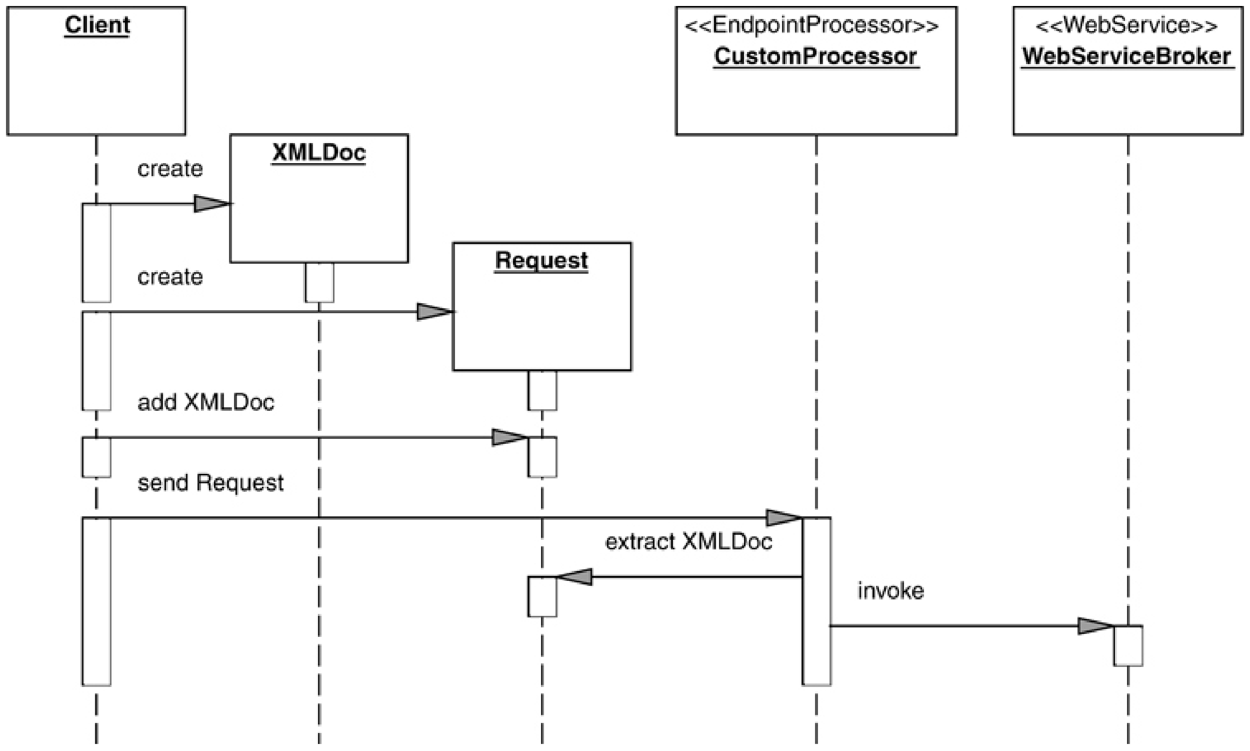
Structure of the web service agent pattern

Publishing and invoking… Web Service Broker



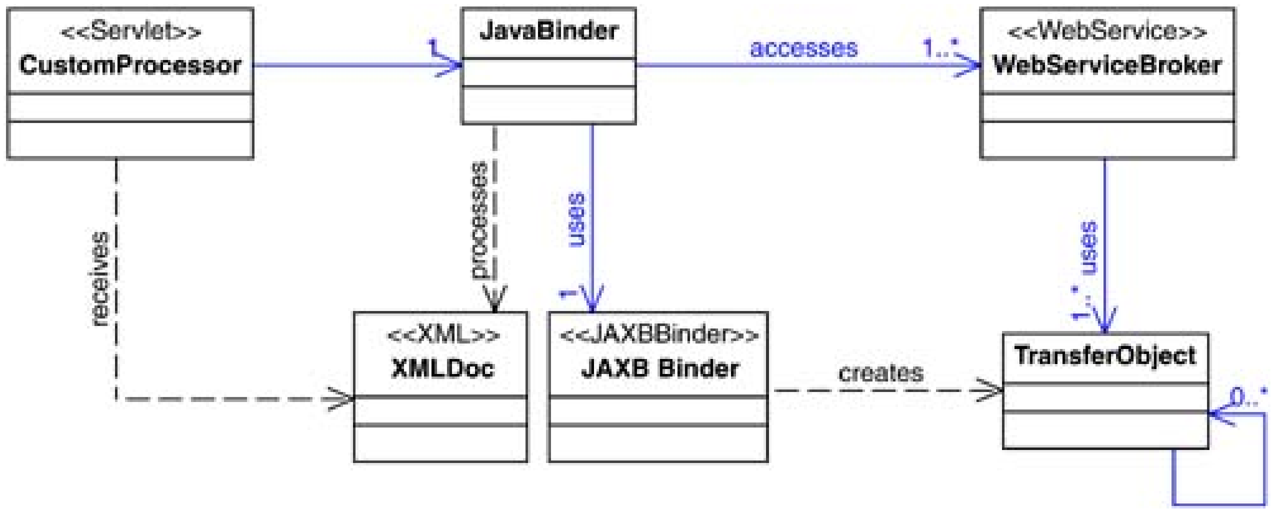
Interaction in the web service agent pattern

Publishing and invoking… Web Service Broker



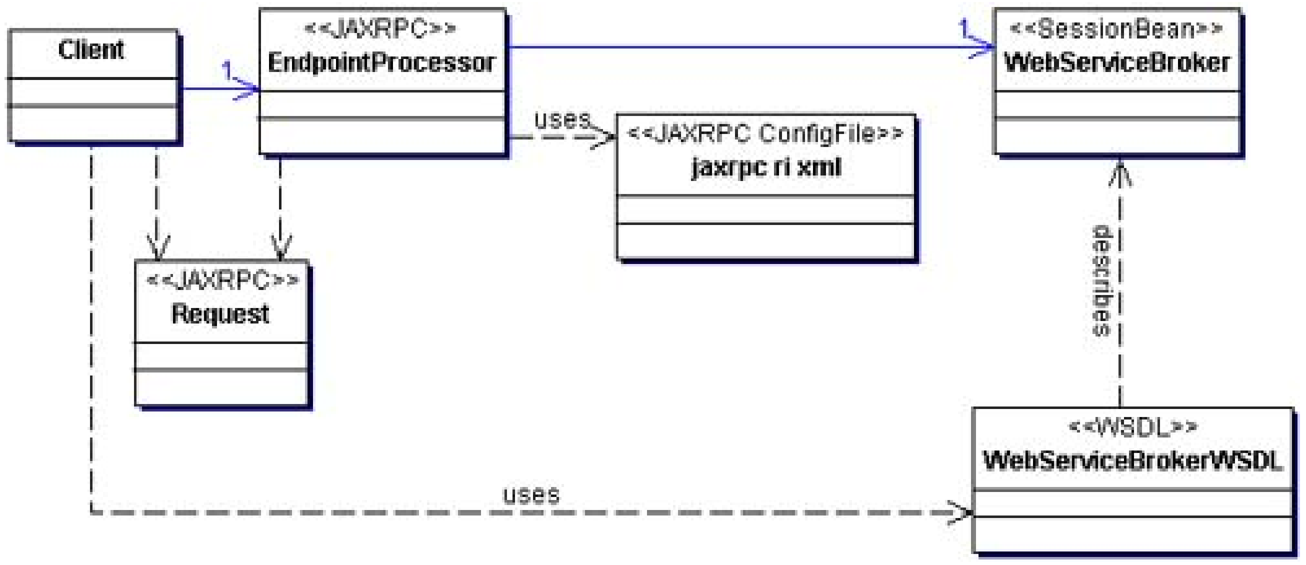
Interaction in the web service agent pattern

Publishing and invoking… Web Service Broker



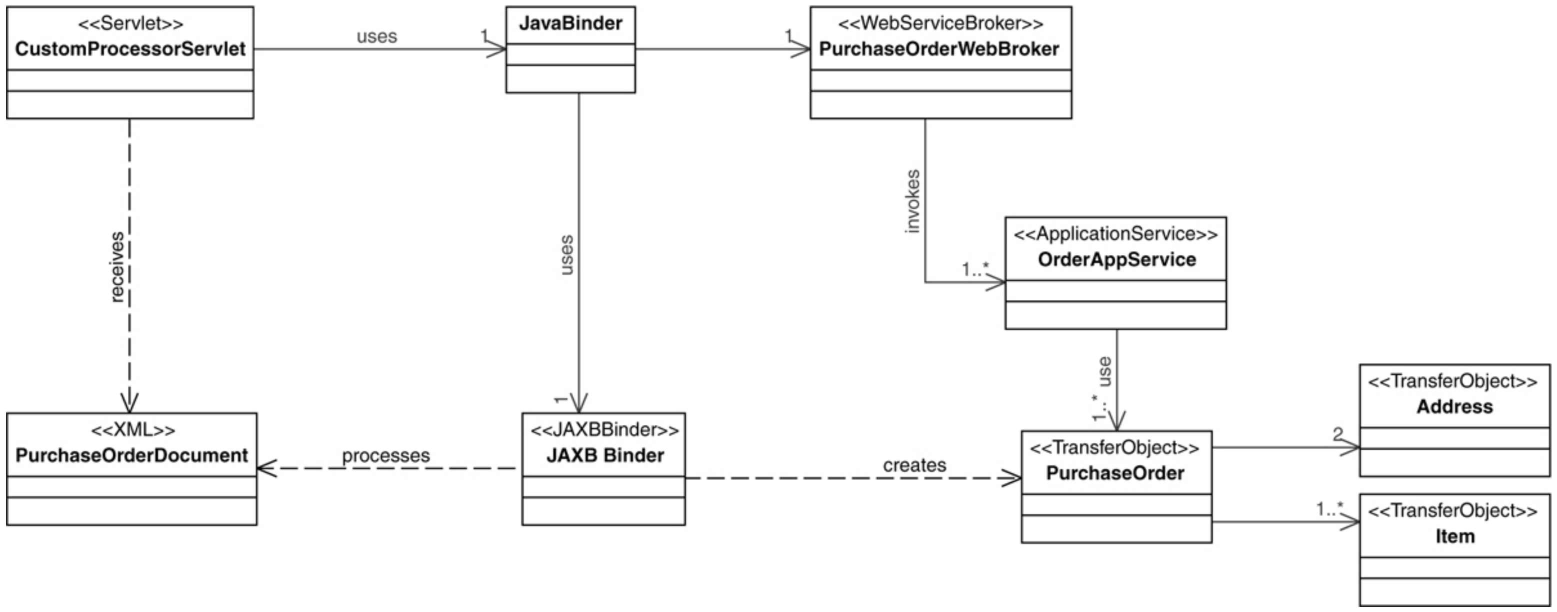
RESTful web services pattern structure

Publishing and invoking… Web Service Broker



Structure of the SOAP web services pattern

Publishing and invoking… Web Service Broker



RESTful Web Service Agent Example

Publishing and invoking… Web Service Broker

- Consequences
  - Advantages
    - Introduces a layer between customer and service

Publishing and invoking… Web Service Broker

• Example:

```
@WebService(targetNamespace = "http://wsb.saludo.negocio/", portName =
"SaludoWSBPort", serviceName = "SaludoWSBService")
public class SalutationWSB {

  public String greet(String first name, String last name)
    {
      return FactoriaNegocio. getInstancia(). newGreeting().saludar(nombre,
apellido);
    }
```

Publishing and invoking... Web Service Broker

```
public interface Salutation {
    public String greet(String first name, String last name);
}

public class SaludoImp implements Saludo {

    public String greet(String first name, String last name)
    {
        return "Hello "+firstname+""+lastname;
    }

}
```

Publication and… Business Delegate

- Purpose
  - Prevents clients from having to deal with distributed component access details in a multi-tier application
- Also known as:
  - *Business delegate*

Publication and… Business Delegate

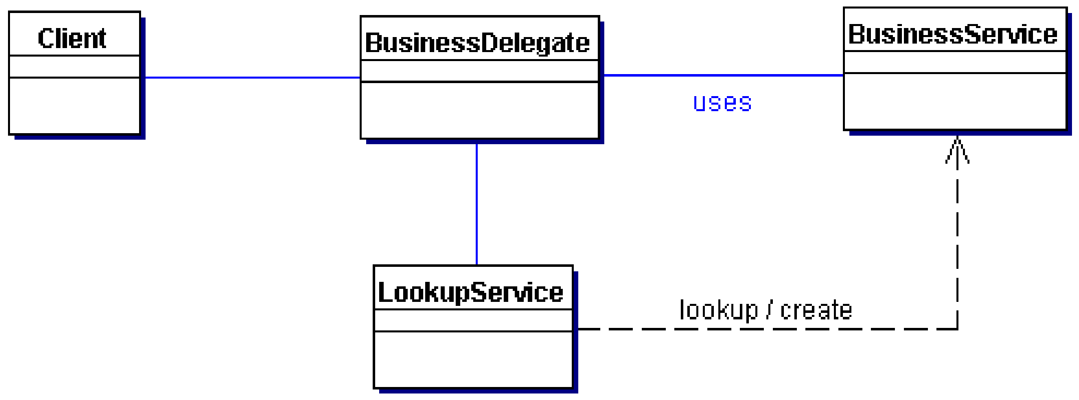
- Motivation
  - When customers interact with remote business services, a variety of problems can occur:
    - Customer and Service Matching
    - Network performance by multiple invocations
    - Details of access to the service
- Context
  - You want to access remote business components
  - You want to minimize the coupling between clients and remote components.
  - You want to avoid unnecessary remote invocations.

Publication and… Business Delegate

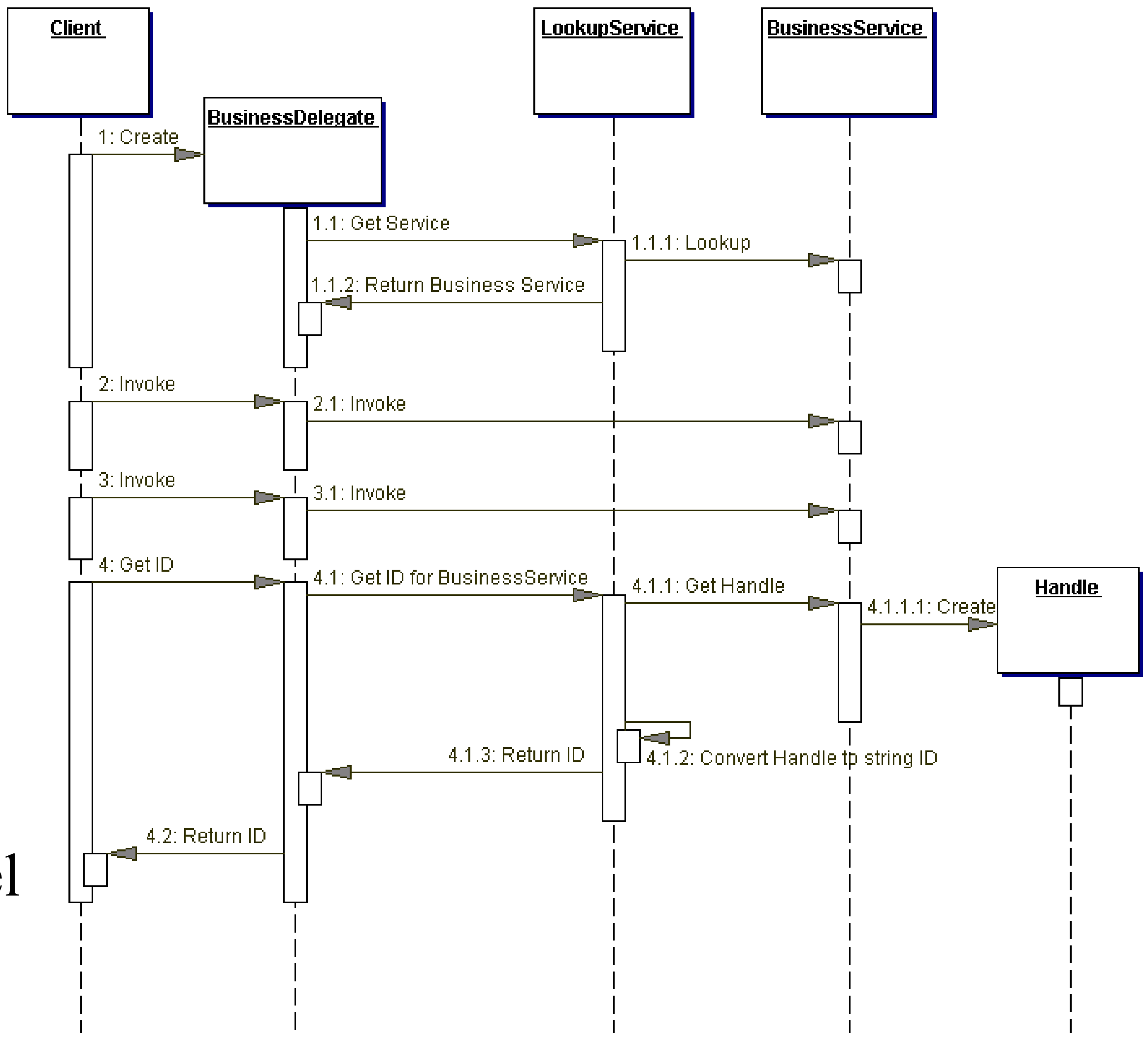
  - You want to translate network exceptions into application or user exceptions.
  - You want to hide the details of service creation, reconfiguration and invocation attempts from clients
- Solution
  - Using a business delegate to encapsulate access to remote business services
  - The business delegate hides business service implementation details, such as search and access mechanisms.

Publication and… Business Delegate

- Description



Delegated Business Pattern Structure



Interaction in the delegated business model

Publication and… Business Delegate

- Consequences
  - Advantages
    - Reduces coupling, improves modularity
    - Translates business service exceptions
    - Improved availability
    - Exposes a uniform and simpler interface to the business layer
    - Improved performance
  - Disadvantages
    - Enter an additional level
    - Hide the location of remote services

Publication and… Business Delegate

- Example code:

```
public abstract class DelegateGreeting {
  protected static DelegateGreeting instance;

  public static DelegateGreeting getInstance()
  {
    if (instance==null) instance= new DelegateGreetingImp();
    return instance;
  }
abstract public String greet(String first name, String last name);
```

Publication and… Business Delegate

```
public class DelegateGreetingImp extends DelegateGreeting {
  protected Saludo port;
  public DelegateGreetingImp() {
    try {
          QName SERVICE_NAME = new QName("http://wsb.saludo.negocio/",
"SaludoWSBService");
          URL wsdlURL = new URL("http://localhost:8080/saludoSOAP/wsdl/saludowsb.wsdl");

          Service ss = Service. create(wsdlURL, SERVICE_NAME);
          port= ss.getPort(business.greeting.imp.greeting. class);

      } catch (Exception e) {e.printStackTrace(); } } }
```

Publication and... Business Delegate

```
public String greet(String first name, String last name)
{
  return port.greet(firstname, lastname);
}
```

JAX-WS. Introduction

- JAX-WS makes it easy to define and use web services
- Using Java `@anotations`, automatic tools can publish the WSDL interfaces necessary for their use.
- It is used together with JAXB (*Java Architecture for XML Bind*), which maps Java classes to XML schemas and vice versa.

JAX-WS. Service annotations

- On the side that implements the service there are a number of key annotations:
  - `@WebService`: indicates the Java class responsible for implementing the web service. It can be used directly on classes or interfaces
  - `@WebMethod`: indicates that a Java method is exposed as a web service operation
  - `@WebParam`: customize the parameter of a web service method to the corresponding WSDL message part
  - `@WebResult`: specifies the mapping of the return type of a web service method
  - `@SOAPBinding`: specifies mapping the web service to SOAP message protocol

JAX-WS. Service annotations

- Example

```
@WebService(targetNamespace = "http://wsb.saludo.negocio/", portName =
"SaludoWSBPort", serviceName = "SaludoWSBService")
public class SalutationWSB {
  public String greet(String first name, String last name)
  {
     return FactoriaNegocio. getInstancia(). newGreeting().saludar(nombre,
apellido);
  }
}
```

JAX-WS. Service annotations

- Particularly important are the attributes of `@WebService`:
  - `name`: specifies the name of the service interface (by default, the class name). It is the value of the `wsdl:portType` attribute in the WSDL contract.
  - `targetNamespace`: specifies the namespace
  - `serviceName`: specifies the name of the published service. This is the value of the attribute of the `wsdl:service` element in the WSDL contract.
  - `wsdlLocation`: specifies the URI where the WSDL contract is located. By default this is where the service is deployed.
  - `endpointInterface`: specifies the name of the *Service Endpoint Interface* (SEI) that implements the exposed class as a web service (if any).
  - `portName`: specifies the name of the *end point* where the service is published. It is the value of the `wsdl:port` attribute in the WSDL contract.

JAX-WS. Service annotations

```
<? xml version="1.0" encoding="UTF-8"?>

< wsdl:definitions name="SalutationWSBService"
targetNamespace="http://wsb.saludo.negocio/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://wsb.saludo.negocio/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
```

JAX-WS. Service annotations

```
   < wsdl:portType name="SalutationWSB">
    < wsdl:operation name="greet">
      < wsdl:input name="greet" message="tns:greet">
    </wsdl:input>.
      < wsdl:output name="greetResponse" message="tns:greetResponse">
    </wsdl:output>.
    </wsdl:operation>
  </wsdl:portType>
```

JAX-WS. Service annotations

```
< wsdl:binding name="GreetingWSBServiceSoapBinding" type="tns:GreetingWSB">
  < soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  < wsdl:operation name="greet">
    < soap:operation soapAction="" style="document"/>
    < wsdl:input name="greet">
      < soap:body use="literal"/>
    </wsdl:input>.
    < wsdl:output name="greetResponse">
      < soap:body use="literal"/>
    </wsdl:output>.
  </wsdl:operation>
  </wsdl:binding>
```

JAX-WS. Service annotations

```
.......................
< wsdl:service name="SaludoWSBService">
    < wsdl:port name="SalutationWSBPort" binding="tns:SalutationWSBServiceSoapBinding">
      < soap:address
location="http://localhost:8080/saludoSOAP/services/SaludoWSBPort"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

JAX-WS. Service annotations

- In CXF the web service publication is declared in the file `cxf-beans.xml`, which can contain more than one service in a project

```
<? xml version="1.0" encoding="UTF-8"?>
< beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jaxws="http://cxf.apache.org/jaxws"
       xsi:schemaLocation=
               "http://www.springframework.org/schema/beans
               http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
               http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">
```

JAX-WS. Service annotations

```
<!- If an SEI is used in a different package than the service, CXF puts in xmln:tns
the SEI's namespace, instead of the service's, and generate an error in Tomcat -->.
< jaxws:endpoint xmlns:tns="http://wsb.saludo.negocio/"
                      id="saludowsb"
                      implementor="business.greeting.wsb.greetingWSB"
                      wsdlLocation="wsdl/saludowsb.wsdl"
                      endpointName="tns:SalutationWSBPort".
                      serviceName="tns:SalutationWSBService"
                      address="/SalutationWSBPort">
               < jaxws:features>
                  < bean class="org.apache.cxf.feature.LoggingFeature" />
               </jaxws:features>
    </jaxws:endpoint>.
</beans>
```

JAX-WS. Service annotations

- It is worth discussing whether or not it is necessary to define a *Service Endpoint Interface*, SEI.
  - Since the WSDL contract itself plays the role of an interface, it is not necessary to
  - Perhaps it is more appropriate to define web service in the context of a *web service broker.*
  - In any case, if you publish an SEI, you need to publish all its operations. In the case of a class, you do not need to

JAX-WS. Customer notes

- On the client side there are also necessary annotations
- In this case it is necessary to have an SEI, at least annotated with the values of `targetNamepace` and name

```
@WebService(targetNamespace = "http://wsb.saludo.negocio/", name = "SaludoWSB")
  public interface Salutation {
    public String greet(String first name, String last name);
```

JAX-WS. Customer notes

- Otherwise, we simply need to instantiate the `javax.xml.ws.Service` class, which is the JAX-WS SOAP access proxy factory.
- The proxy needs the access data to the web service implementation, which is provided with an instance of `java.xml.namespace.Qname`, with the values of the service provider's `targetNamespace` and `serviceName`. You also need the access URL to the WSDL contract
- This proxy instantiates the SEI defined in the client, from the WSDL contract.

JAX-WS. Customer notes

```
QName SERVICE_NAME = new QName("http://wsb.saludo.negocio/", "SaludoWSBService");
URL wsdlURL = new URL("http://localhost:8080/saludoSOAP/wsdl/saludowsb.wsdl");
Service ss = Service. create(wsdlURL, SERVICE_NAME);
Greeting port= ss.getPort(main.Greeting. class);

response= port.greet(firstname, lastname);
```

JAX-RS. Introduction

- JAX-RS facilitates both the invocation and response processing of REST web services.
- Like JAX-WS, it uses Java `@anotations`, to generate the REST web service implementation.
- It is used together with JAXB (*Java Architecture for XML Bind*), which maps Java classes to XML schemas and vice versa.

JAX-RS. Service annotations

- In this case, there are only annotations on the side that implements the service:
  - `@Path`: identifies the template URI that gives access to the service
  - `@xParam`: parameters extracted from the URI via patterns
  - `@POST`, `@GET`, `@PUT`, `@DELETE`: identify the HTTP access methods, which invoke precisely the methods of the classes they annotate.
  - `@Consumes`: MIME media type received by a web service
  - `@Produces`: MIME media type produced by a web service

JAX-RS. Service annotations

```
@Path("/greeting/wsb")
public class SalutationWSB {

  @GET
    @Path("{firstname}/{lastname}")
    @Produces("text/plain")
  public String read(@PathParam("firstName") String firstName,
                    @PathParam("lastName") String lastName)
  {
      return FactoriaNegocio. getInstancia().
nuevoSaludo().saludar(nombre+""+apellido);
  }
```

JAX-RS. Service annotations

- The linking of the input URLs with the classes is done through the `web.xml` file, which can define several services including several `servlet` definitions (unique name) and several `servlet-mapping`

```
<? xml version="1.0" encoding="UTF-8"?>
< web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
```

JAX-RS. Service annotations

```
< servlet>
    < servlet-name> CXFServlet</servlet-name>
    < servlet-class> org.apache.cxf.jaxrs.servlet.CXFNonSpringJaxrsServlet
    </servlet-class>
    < init-param>
       < param-name> jaxrs.serviceClasses</param-name>.
       < param-value>business.greeting.wsb.GreetingWSB</param-value>.
    </init-param>
    <load-on-startup> 1</load-on-startup>.
 </servlet>
```

JAX-RS. Service annotations

```
    < servlet-mapping>
      < servlet-name> CXFServlet</servlet-name>
      < url-pattern> /services/*</url-pattern>
  </servlet-mapping>.

</web-app>
```

JAX-RS. Service annotations

- JAX-RS is quite flexible with respect to the @Path annotation, so the following annotations correspond to the same URL:

```
http://localhost:8080/saludoREST/servicios/saludo/wsb/Charlton/Heston
```

```
@Path("/greeting/wsb")
public class SalutationWSB {
  @GET
    @Path("/{firstname}/{lastname}") .............................

@Path("/greeting/wsb")
public class SalutationWSB {
  @GET
    @Path("{firstname}/{lastname}") .............................
```

JAX-RS. Service annotations

- It is worth mentioning that JAX-RS can manage different types of parameters:
  - `@PathParam`: patterns identified in the URL. For example, `http://localhost:8080/saludoREST/servicios/saludo/wsb/Charlton/Heston`
  - `MatrixParam`: value name pairs in the URL, preceded by ";". For example, `http://servidor.es/producto/crear/27;active=false`
  - `@QueryParam`: traditional parameters in URLs. For example: `http://localhost:8080/saludoREST/saludo/wsb?nombre=Miguel&apellido=Ríos`
  - `@FormParam`: parameters coming from forms

JAX-RS. Service annotations

- `@HeaderParam`: parameters included in the HTTP header
- `@CookieParam`: parameters included in cookies
- `@BeanParam`: classes built automatically from parameters. For example,

```
Public class CustomerInput {
      @FormParam("first")
      String firstName;
      @FormParam("last")
      String lastName;
            @HeaderParam("Content-Type")
      StringContentType;
      public String getFirstName() { return firstName, }
……………..  }
```

JAX-RS. Service annotations

```
      @Path("/customers")
      public class CustomerResource {

            @POST
            public void createCustomer(@BeanParam CustomerInput newCust)
{ ........}
      …………………………………………………….
}
```

JAX-RS. Service annotations

- The service can also receive information in XML or JSON format, which:
  - It can be sent directly in these formats
  - It can come from objects that can be serialized to those formats with additional frames such as:
    - JAXB for XML
    - JSON-B for JSON
      - You have to configure CXF with a `cxf.xml` file as it says (Javatips.net)
      - (JAX-RS Data Bindings) also gives more information about it

JAX-RS. Client API

- JAX-RS makes client-side invocation quite easy thanks to the API provided by
- There are several key classes:
  - `ClientBuider`: API entry point when creating `Client` instances
  - `Client`: API entry point for constructing and executing client requests and for consuming the returned responses
  - `WebTarget`: target resource identified by URL
  - `Invocation.Builder`: the client request builder. It is the one that sends `GET`, `POST`, `PUT`, `DELETE` requests.
  - `Response`: response provided by `GET`, `POST`, `PUT`, `DELETE` invocation. Can be translated directly to other types supported by `Entity`
  - `Entity`: information sent in request and obtained in response. For example: `APPLICATION_JSON`, `APPLICATION_SVG_XML`, `APPLICATION_XML`, `TEXT_HTML`, `TEXT_PLAIN`

JAX-RS. Client API

- For example:

```
String url= "http://localhost:8080/saludoREST/servicios/saludo/wsb";
Client client = ClientBuilder. newClient();

String res= client.target(url + "/Charlton/Heston").request().get(String. class);

System. out.println(res);
client.close();
```

JAX-RS. Client API

- The `get()`, `post()`, `put()` and `delete()` methods of `Invocation.Builder` return a `Response.`
- In the example above, the client can directly consume a `String`, because the methods allow you to set the response type to a different one (the framework handles the conversion from the `Response` type to the specified one).

JAX-RS. Example

- Let's look at a simple example involving all four methods
- Let's look at it by invocation/service pairs:

```
String url= "http://localhost:8080/saludoREST/servicios/saludo/wsb";
Client client = ClientBuilder. newClient();
String res= client.target(url + "/Charlton/Heston").request().get(String. class);
System. out.println(res);
```

JAX-RS. Example

```
@Path("/greeting/wsb")
public class SalutationWSB {

  @GET
    @Path("{firstname}/{lastname}")
    @Produces("text/plain")
  public String read(@PathParam("firstName") String firstName, @PathParam("lastName")
String lastName)
  {
    return FactoriaNegocio. getInstancia(). nuevoSaludo().saludar(nombre+""+apellido);
  }
```

JAX-RS. Example

```
res= client.target(url). request().post(Entity. text("Gwyneth, Paltrow"), String.
class);

System. out.println(res);


  @POST

  public Response readPOST(String s)

        { List<String> l = Arrays. asList(s.split("\s*,\s*"));

      String name= l.get(0);

      String lastname= l.get(1);

      String res= FactoriaNegocio. getInstancia(). nuevoSaludo().saludar(nombre+" "
+apellido+"-POST");

     return Response. ok(res). build();
```

JAX-RS. Example

```
res= client.target(url + "?firstname=Miguel&lastname=Rios"). request(). delete(String.
class);
System. out.println(res);


  @DELETE
    @Produces("text/plain")
  public String readDELETE(@QueryParam("firstName") String firstName,
@QueryParam("lastName") String lastName)
  {
        return FactoriaNegocio. getInstancia().
nuevoSaludo().saludar(nombre+""+apellido+"-DELETE");
```

JAX-RS. Example

```
Form form= new Form();
form.param("name", "Melania");
form.param("last name", "Trump");

res= client.target(url).request().put(Entity. form(form), String. class);

System. out.println(res);
```

JAX-RS. Example

```
@PUT
public String readPUT(@FormParam("firstName") String firstName,
                      @FormParam("lastName") String lastName)
{

    String res= FactoriaNegocio. getInstancia(). nuevoSaludo().saludar(nombre+" "
+apellido+"-PUT");


    return res;
```

JAX-RS. Example

```
Client client2 = ClientBuilder. newClient().
register(org.eclipse.yasson.JsonBindingProvider. class);
String url2= "http://localhost:8080/saludoREST/servicios/saludo/wsb/json";


Persona p= new Persona("Pepe", "Romero");
Greeting s= client2.target(url2). request(MediaType. APPLICATION_JSON)
                                 . put(Entity.json(p), Greeting. class);


System. out.println(s.getText());
```

JAX-RS. Example

```
@XmlRootElement
public class Person {
        protected String name;
        protected String last name;
……………..}

@XmlRootElement
public class Salutation {
        String text;
……..}
```

JAX-RS. Example

```
@PUT
@Path("/json")
  @Consumes(MediaType. APPLICATION_JSON)
  @Produces(MediaType. APPLICATION_JSON)
public Greeting readPUTjson(Person p)
{
      Greeting s= new Greeting();
  s.setText(BusinessFactory. getInstance(). newGreeting().greeting(p.getFirstName()+"
" +p.getLastName()+"-PUTjson"));

  return s;
```

Conclusions

- JAX-WS and JAX-RS make it much easier to implement Java web services.
- JAX-WS is quite convenient when used in conjunction with the IDE
- Which one is better? I don't know
- Which one is simpler? I don't know, but JAX-WS has been seen in 14 slides and JAX-RS in 23 slides.

Java Web Services

Intelligent Infrastructure Design for the Internet of Things

Antonio Navarro