Web Services

Daniel Hagimont

IRIT/ENSEEIHT
2 rue Charles Camichel - BP 7122
31071 TOULOUSE CEDEX 7

Daniel.Hagimont@enseeiht.fr
http://hagimont.perso.enseeiht.fr

Motivations

- Motivations
  - Coarse-grained application integration
  - Unit of integration: the "service" (interface + contract)
- Constraints
  - Applications developed independently, without anticipation of any integration
  - Heterogeneous applications (models, platforms, languages)
- Consequences
  - No definition of a common model
  - Elementary common basis
    - For communication protocols (messages)
    - For the description of services (interface)
  - Base choice: XML (because of its adaptability)

Web Services (WS)

- Conceptual contribution
  - No new fundamental concept …
  - … so, what for ?
- Concrete contribution
  - Practically address the heterogeneity problem
  - Large-scale (world wide) integration of application
  - Heavy implication of main IT actors

Basic form of WS : XML-RPC

Description in XML of a remote procedure call
Parameter types are specified in an XML schema

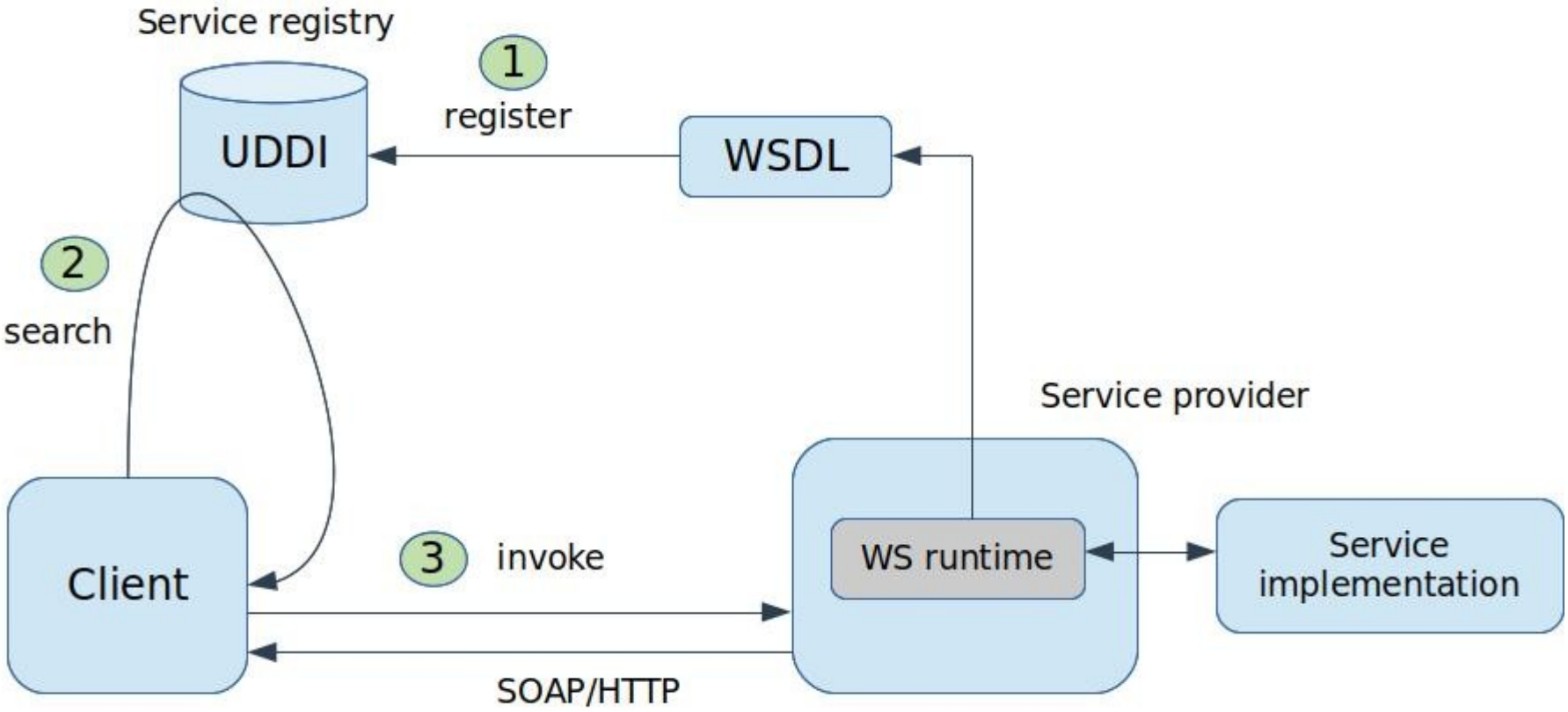
```
<methodCall>
   <methodName>meteo.temperature</methodName>
   <params>
      <param>
         <value><int>31130</int></value>
      </param>
   </params>
</methodCall>
```

Description in XML of parameter retuns

```
<methodResponse>
   <params>
      <param>
         <value><int>25</int></value>
      </param>
   </params>
</methodResponse>
```

Interest : independence with respect to platforms and communication protocols

Architecture of WS



Elements of WS

- Description of a service
  - WSDL : Web Services Description Language
  - Standard notation for the description of a service interface
- Access to a service
  - SOAP : Simple Object Access Protocol
  - Internet protocol allowing communication between Web Services
- Registry of services
  - UDDI : Universal Description, Discovery and Integration
  - Protocol for registration and discovery of services

Tools

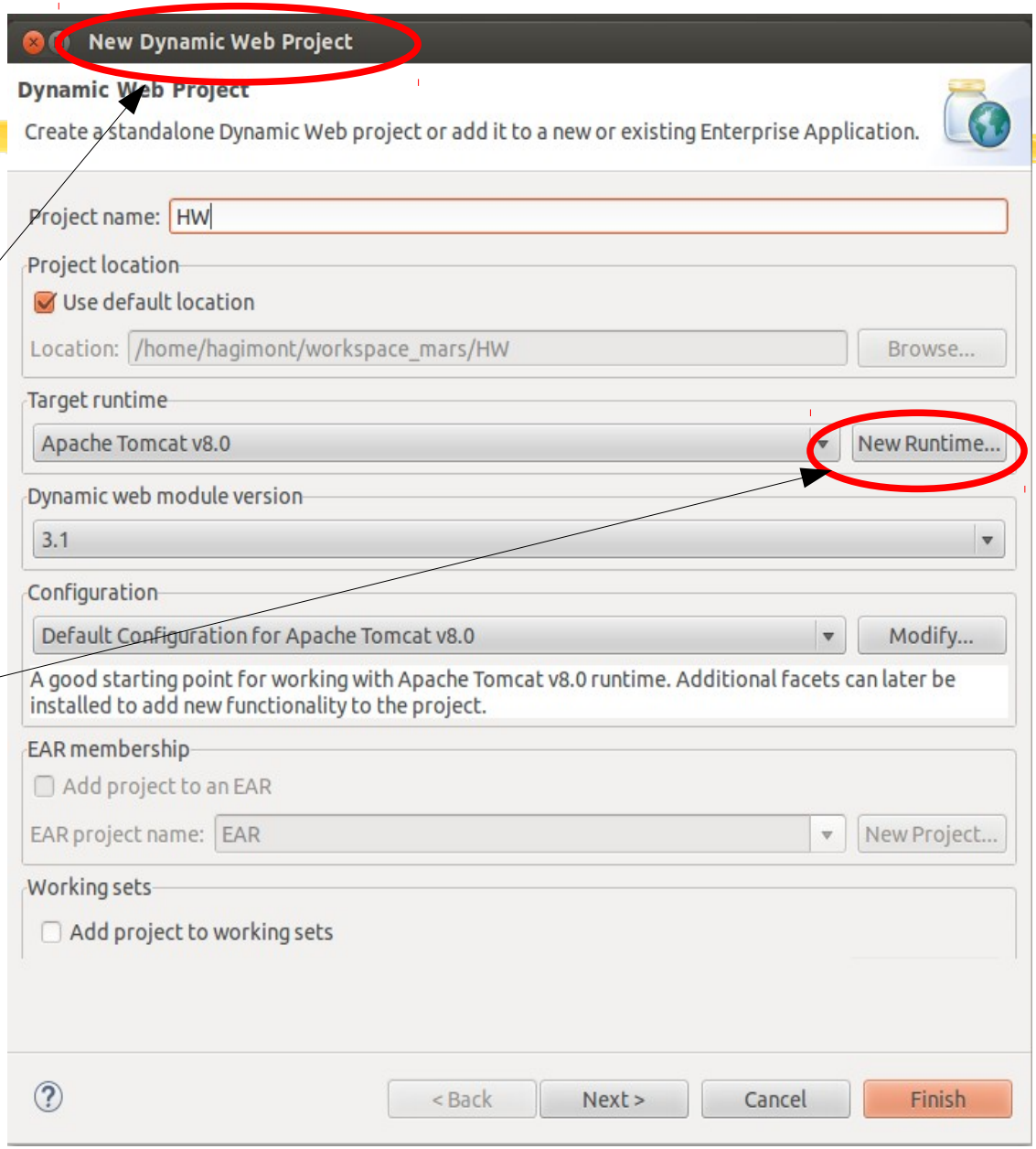
- From a program, we can generate a WS skeleton
  - Example: from a Java program, we generate
    - A servlet which receives SOAP/HTTP requests and reproduces the invocation on an instance of the class
    - A WDSL file which describes the WS interface
- The generated WSDL file can be given to clients
- From WSDL file, we can generate a WS stub
  - Example: from a WSDL file, we generate Java classes which can be used to invoke the remote service
- Programming is simplified
- Such tools are available in different langage environments

Example: programming a Web Services

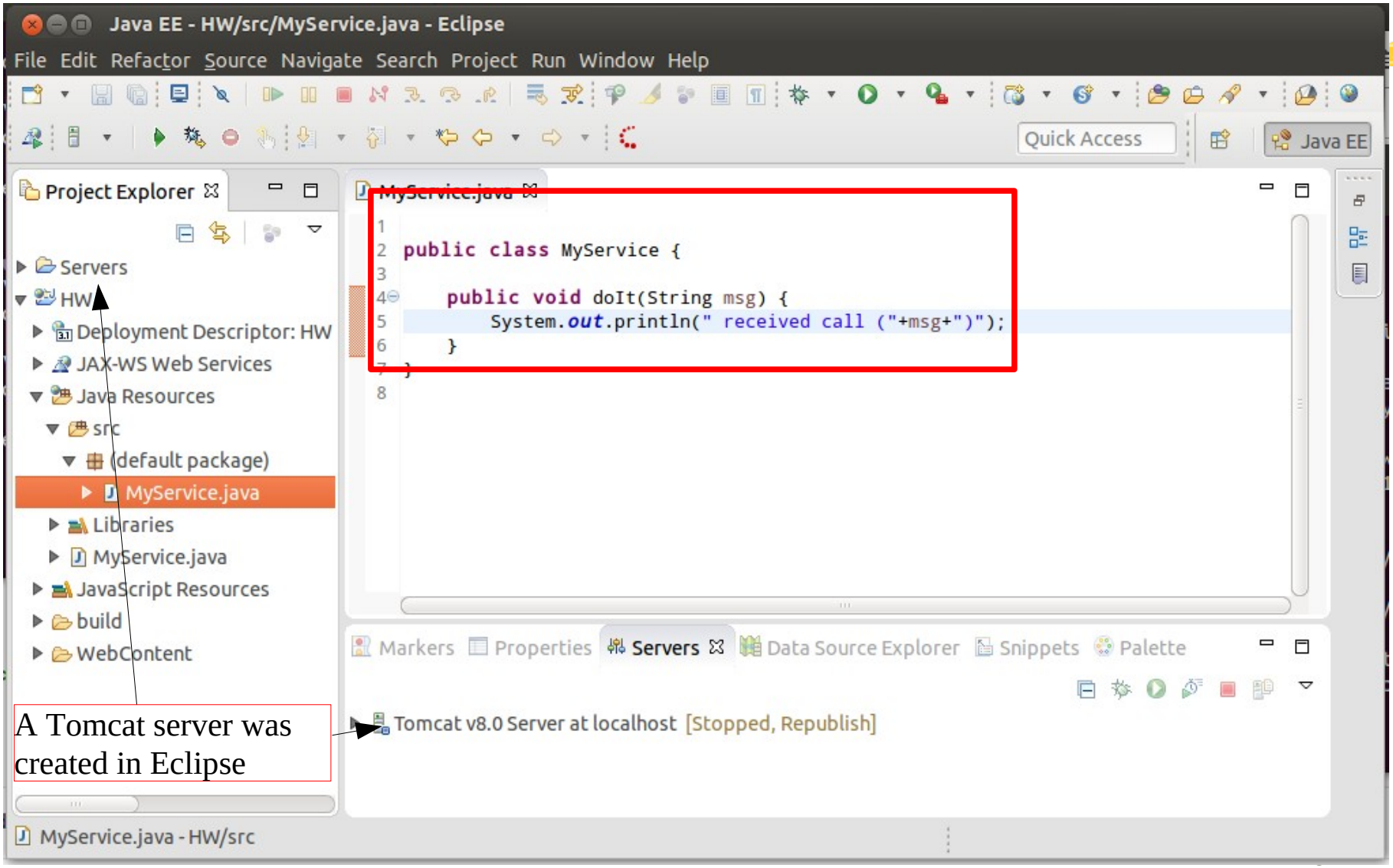
- Eclipse JEE
- Apache Axis
- Creation of a Web Service
  - From a Java class
  - In the Tomcat runtime
  - Generation of the WSDL file
- Creation of a client application
  - Generation of stubs from a WSDL file
  - Programming of the client

Create a Dynamic Web Project

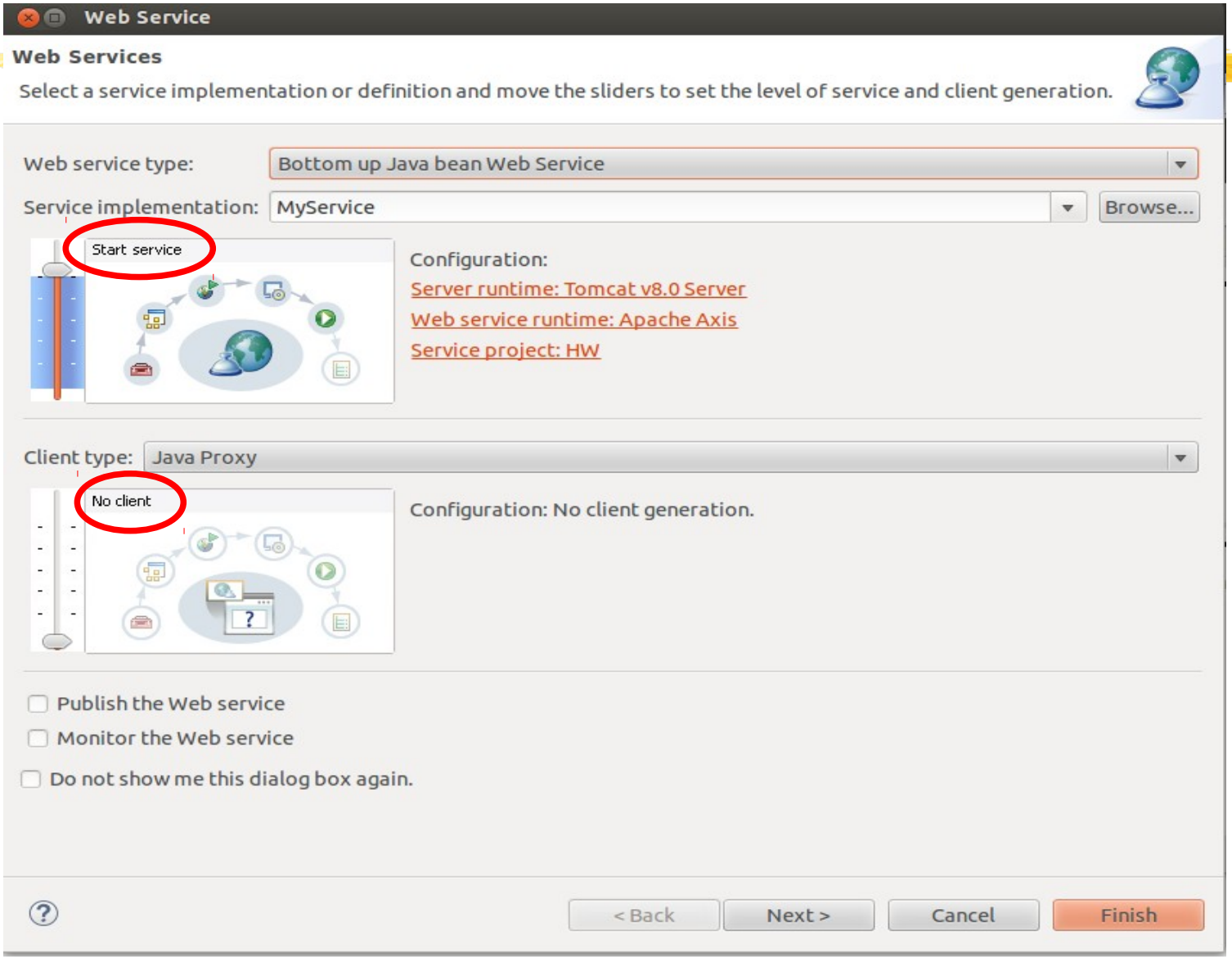
- Eclipse JEE
- Open JEE perspective
- Create a Dynamic Web Project
- Add your Tomcat runtime



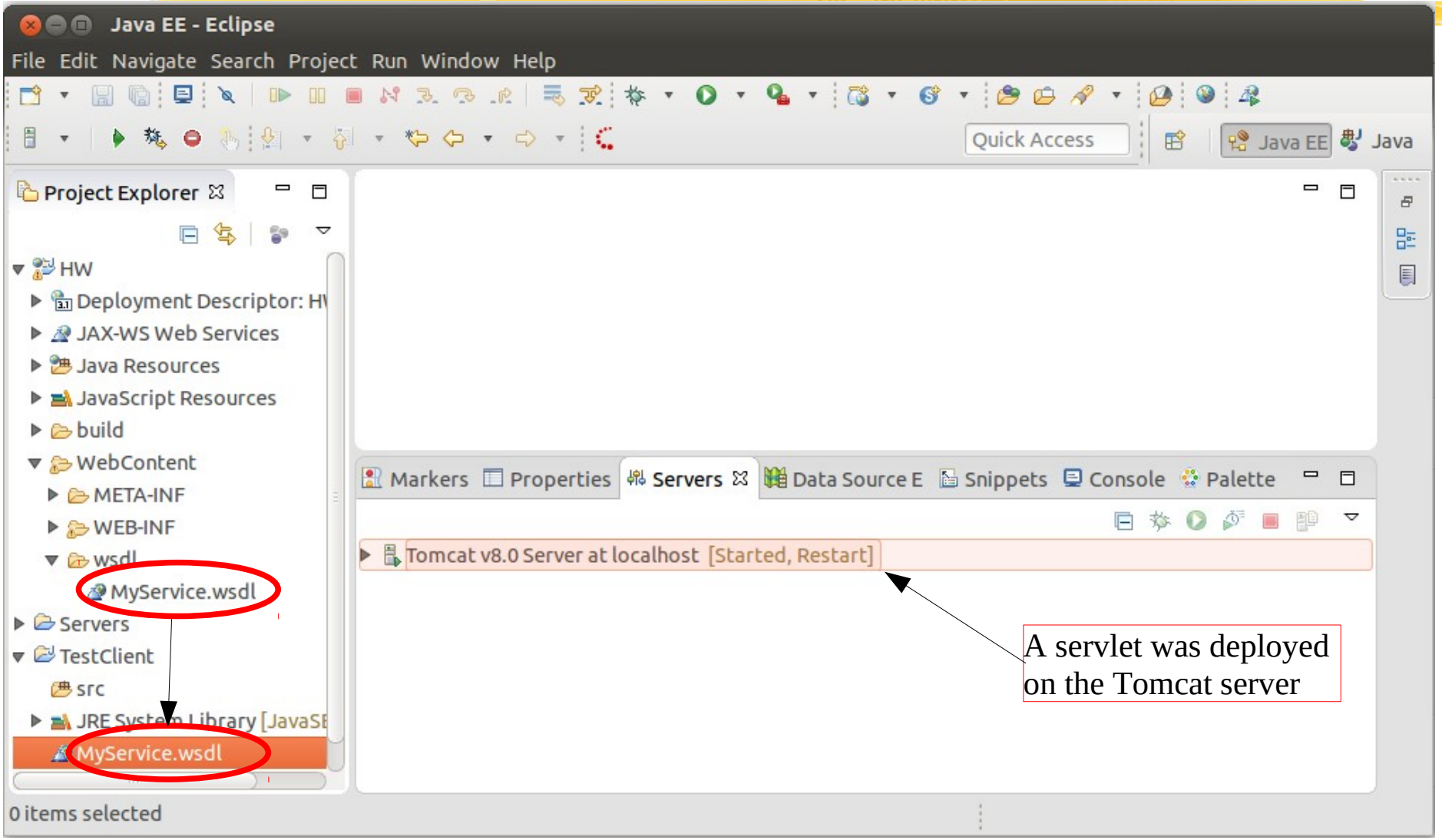
Create a Class



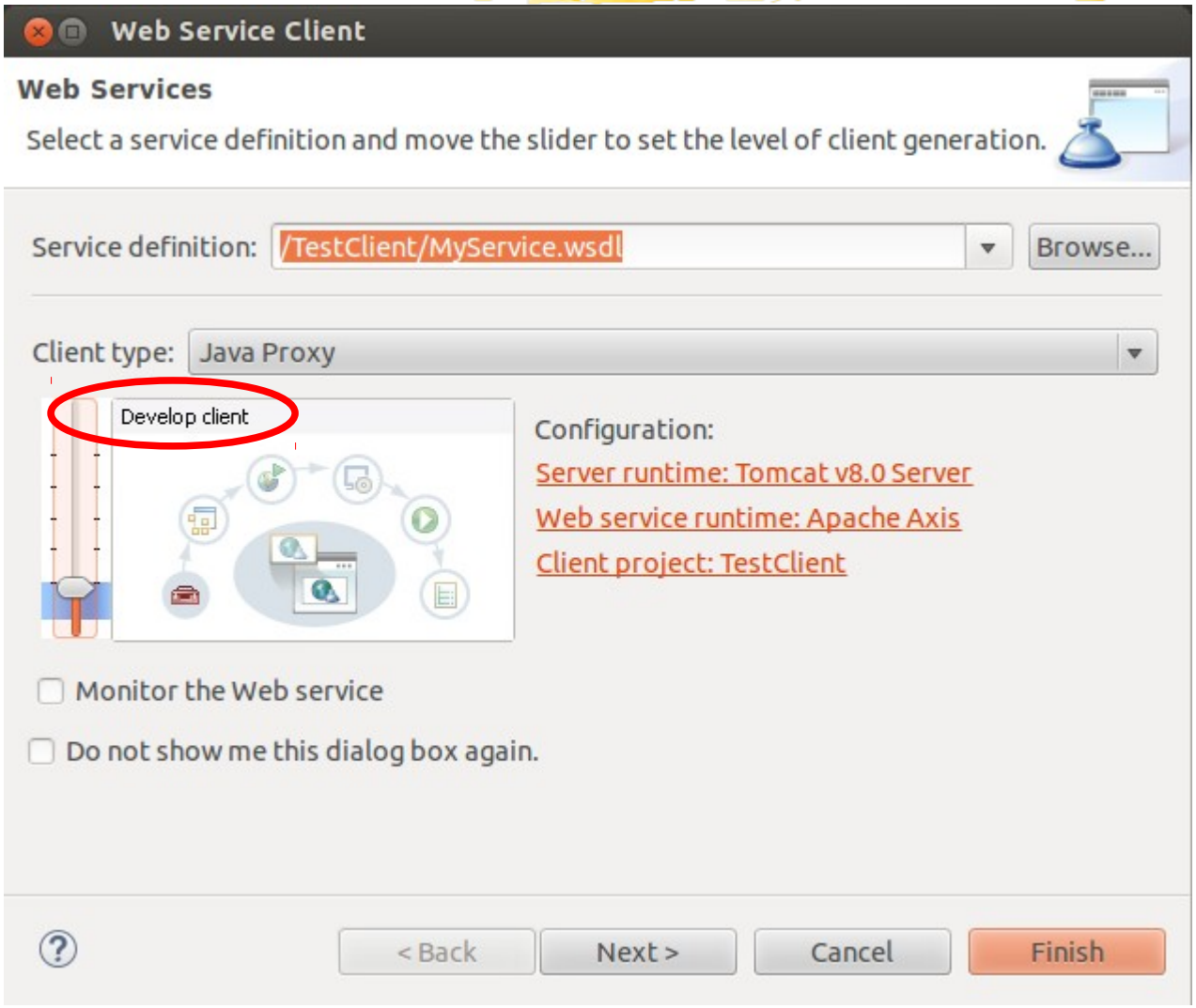
From source file : Web Service → create Web Service



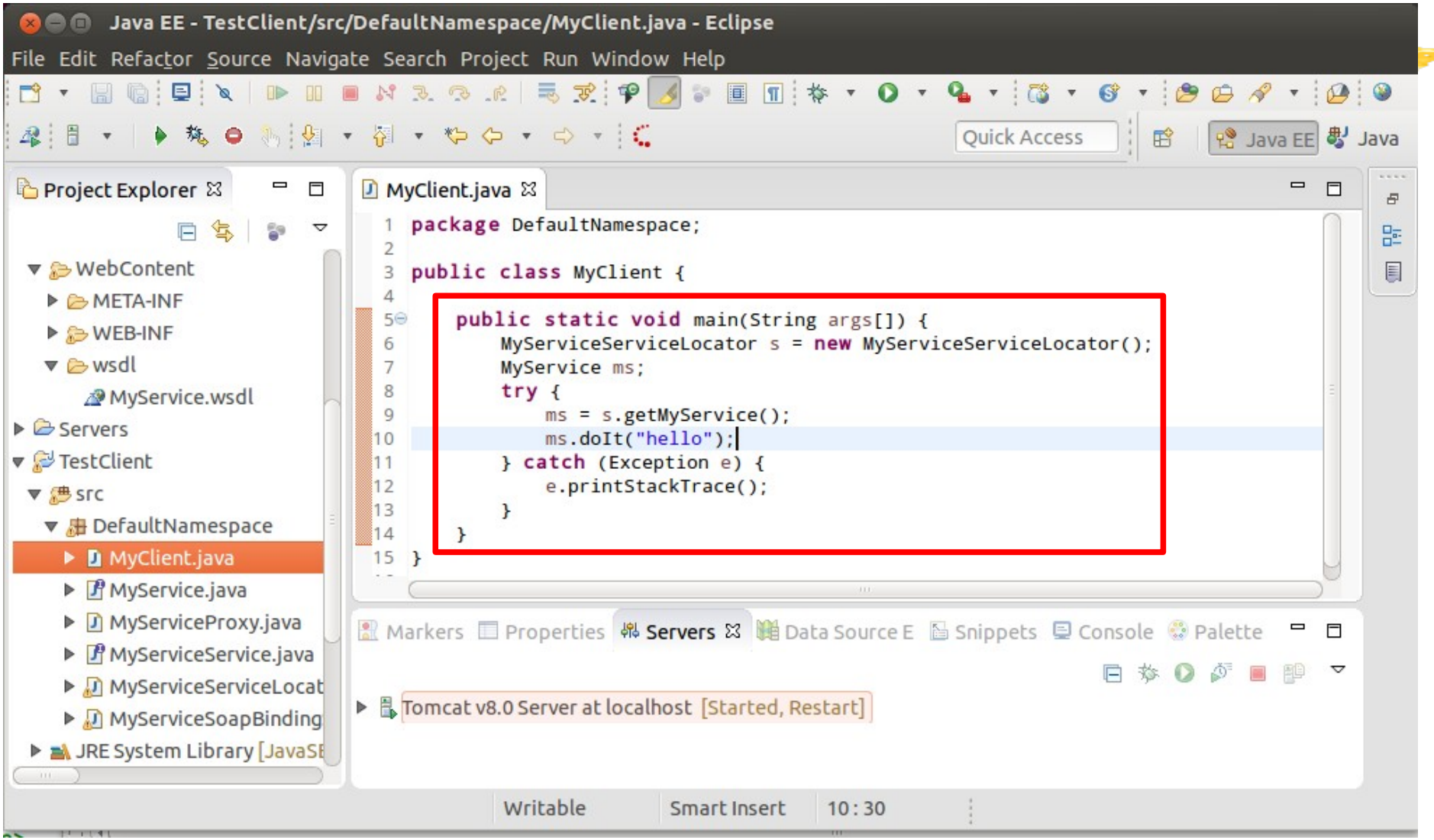
Copy the generated WSDL file in a new Java project



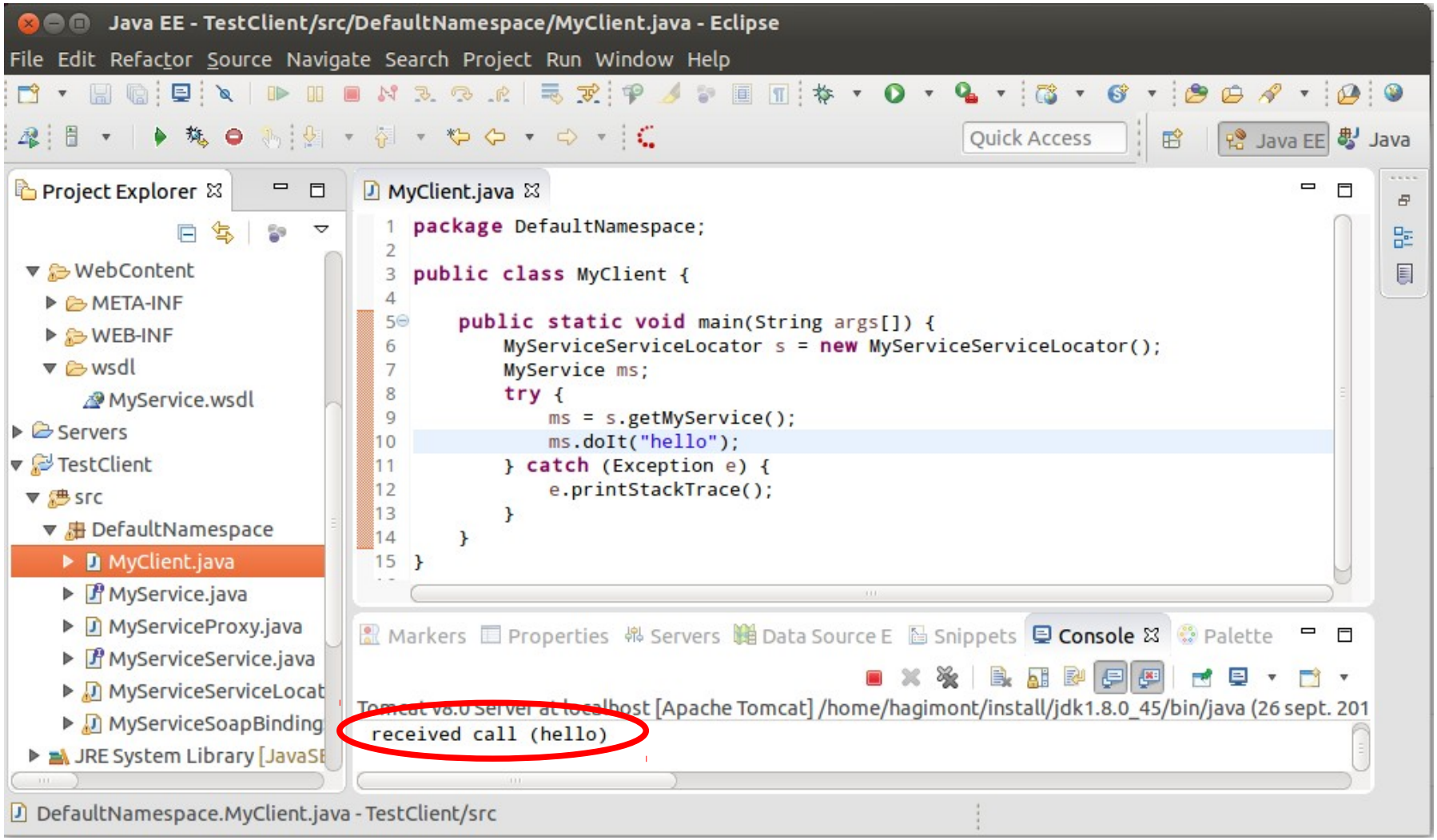
From the WSDL file
Web Service → Generate Client (Develop Client)



Program a client



Run


Java EE - TestClient/src/DefaultNamespace/MyClient.java - Eclipse
File Edit Refactor Source Navigate Search Project Run Window Help
Quick Access
Java EE
Java
Project Explorer
WebContent
META-INF
WEB-INF
wsdl
MyService.wsdl
Servers
TestClient
src
DefaultNamespace
MyClient.java
MyService.java
MyServiceProxy.java
MyServiceService.java
MyServiceServiceLocat
MyServiceSoapBinding
JRE System Library [JavaSE
MyClient.java
1 package DefaultNamespace;
2
3 public class MyClient {
4
5 public static void main(String args[]) {
6 MyServiceServiceLocator s = new MyServiceServiceLocator();
7 MyService ms;
8 try {
9 ms = s.getMyService();
10 ms.doIt("hello");
11 } catch (Exception e) {
12 e.printStackTrace();
13 }
14 }
15 }
Markers
Properties
Servers
Data Source E
Snippets
Console
Palette
Tomcat v8.0 Server at localhost [Apache Tomcat] /home/hagimont/install/jdk1.8.0_45/bin/java (26 sept. 201
received call (hello)
DefaultNamespace.MyClient.java - TestClient/src

Generated WSDL

```
<wsdl:definitions targetNamespace="http://DefaultNamespace"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://DefaultNamespace"
xmlns:intf="http://DefaultNamespace" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
 <wsdl:types>
  <schema elementFormDefault="qualified" targetNamespace="http://DefaultNamespace"
xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="doIt">
    <complexType>
     <sequence>
      <element name="msg" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
   <element name="doItResponse">
    <complexType/>
   </element>
  </schema>
 </wsdl:types>
```

Generated WSDL

```
<wsdl:message name="doItResponse">
    <wsdl:part element="impl:doItResponse" name="parameters">
    </wsdl:part>
  </wsdl:message>
  <wsdl:message name="doItRequest">
    <wsdl:part element="impl:doIt" name="parameters">
    </wsdl:part>
  </wsdl:message>
  <wsdl:portType name="MyService">
    <wsdl:operation name="doIt">
      <wsdl:input message="impl:doItRequest" name="doItRequest">
     </wsdl:input>
      <wsdl:output message="impl:doItResponse" name="doItResponse">
     </wsdl:output>
    </wsdl:operation>
  </wsdl:portType>
```

Generated WSDL

```
<wsdl:binding name="MyServiceSoapBinding" type="impl:MyService">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="doIt">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="doItRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="doItResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="MyServiceService">
    <wsdl:port binding="impl:MyServiceSoapBinding" name="MyService">
      <wsdlsoap:address location="http://localhost:8080/HW/services/MyService"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

SOAP request(with TCP/IP Monitor)

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance">

    <soapenv:Body>
        <doIt xmlns="http://DefaultNamespace">
            <msg>hello</msg>
        </doIt>
    </soapenv:Body>

</soapenv:Envelope>
```

SOAP response

```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <soapenv:Body>
        <doItResponse xmlns="http://DefaultNamespace"/>
    </soapenv:Body>

</soapenv:Envelope>
```

Restful Web Services

- A simplified version
- Invocation with methods HTTP GET, POST, PUT, DELETE
- Parameter passing in XML or JSON
  - Serialization
- Many development environments
  - Examples : resteasy, jersey

Example with Resteasy (server)

- WS class

```
@Path("/")
public class Facade {

    static Hashtable<String, Person> ht = new Hashtable<String, Person>();

    @POST
    @Path("/addperson")
    @Consumes({ "application/json" })
    public Response addPerson(Person p) {
         ht.put(p.getId(), p);
         return Response.status(201).entity("person added").build();
    }

    @GET
    @Path("/getperson")
    @Produces({ "application/json" })
    public Person getPerson(@QueryParam("id") String id) {
         return ht.get(id);
    }

    @GET
    @Path("/listpersons")
    @Produces({ "application/json" })
    public Collection<Person> listPersons() {
         return ht.values();
    }

}
```

Receives a JSON
Deserialized into a Java object
void method

Returns an object
Serialized into a JSON
Receives an id parameter

Person is a simple POJO

Example with Resteasy (server)

- Add the RestEasy jars in Tomcat
- In eclipse
  - Create a Dynamic Web Project
  - Add RestEasy jars in the buildpath
  - Create a package
  - Implement the WS class (Facade + Person)
  - Add a class RestApp

```
public class RestApp extends Application {
          private Set<Object> singletons = new HashSet<Object>();
          public RestApp() {
                    singletons.add(new Facade());
          }
          Public Set<Object> getSingletons() {
                    return singletons;
          }
}
```

Example with Resteasy (server)

- Add a web.xml descriptor in the WebContent/WEB-INF folder

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" version="3.1">
  <display-name>essai-server</display-name>
  <servlet>
    <servlet-name>resteasy-servlet</servlet-name>
    <servlet-class>
            org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher
        </servlet-class>
    <init-param>
      <param-name>javax.ws.rs.Application</param-name>
      <param-value>pack.RestApp</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>resteasy-servlet</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
</web-app>
```

- Export the war in Tomcat

Publish the WS

- Just write a documentation which says that
  - The WS is available at http://localhost:8080/
  - Method addperson with POST receives a person JSON :

```
{
   "firstname":"Alain",
   "lastname":"Tchana",
   "phone":"0102030405",
   "email":"alain.tchana@enseeiht.fr"
}
```

  - Method getperson with GET receives an id and returns a person
  - Method listperson returns a JSON including a set of persons
- A user may use any tool (not only RestEasy)

Example with Resteasy (client)

- From a documentation of REST WS we can write the interface

```
@Path("/")
public interface FacadeInterface {

    @POST
    @Path("/addperson")
    @Consumes({ "application/json" })
    public Response addPerson(Person p);

    @GET
    @Path("/getperson")
    @Produces({ "application/json" })
    public Person getPerson(@QueryParam("id") String id);

    @GET
    @Path("/listpersons")
    @Produces({ "application/json" })
    public Collection<Person> listPersons();

}
```

Example with Resteasy (client)

- And write a class which invokes the WS

```
public class Client {

   public static void main(String args[]) {

       final String path = "http://localhost:8080/rs-server-person/rest";

       ResteasyClient client = new ResteasyClientBuilder().build();
       ResteasyWebTarget target = client.target(UriBuilder.fromPath(path));
       FacadeInterface proxy = target.proxy(FacadeInterface.class);

       Response resp;
       resp = proxy.addPerson(new Person("007","James Bond"));
       System.out.println("HTTP code: " + resp.getStatus()
                                  +"  message: "+resp.readEntity(String.class));
       resp.close();
       resp = proxy.addPerson(new Person("006","Dan Hagi"));
       System.out.println("HTTP code: " + resp.getStatus()
                                  +"  message: "+resp.readEntity(String.class));
       resp.close();

       Collection<Person> l = proxy.listPersons();
       for (Person p : l) System.out.println("list Person: "+p.getId()+"/"+p.getName());

       Person p = proxy.getPerson("006");
       System.out.println("get Person: "+p.getId()+"/"+p.getName());
   }

}
```

Example with Resteasy (client)

- In eclipse
  - Create a Java Project
  - Add RestEasy jars in the buildpath
  - Implement the Java bean that correspond to the JSON
    - Automatic generation with https://www.site24x7.com/tools/json-to-java.html
  - Implement the interface and the client class (FacadeInterface + Client)
  - Run

Example of existing REST WS

- www.amdoren.com
- Currency converter

Currency API Request

The base URL for our currency API is

`https://www.amdoren.com/api/currency.php`

Request Parameters

Parameter	Description
api_key	Your assigned API key. This parameter is required.
from	The currency you would like to convert from. This parameter is required.
to	The currency you would like to convert to. This parameter is required.
amount	The amount to convert from. This parameter is optional. Default is a value of 1.

Currency API Response

Element	Description
error	Error code. Value greater than zero indicates an error. See list below.
error_message	Short decription of the error. See list below.
amount	The exchange rate or amount converted.

Example:
JSON data returned from our currency API request:

```
{ "error" : 0, "error_message" : "-", "amount" : 0.90168 }
```

Example of existing REST WS

Interface

```
@Path("/")
public interface ServiceInterface {

    @GET
    @Path("/currency.php")
    @Produces({ "application/json" })
    public Result convert(@QueryParam("api_key") String key, @QueryParam("from") String
from,
                    @QueryParam("to")String to);

}
```

Java bean (from JSON)

```
public class Result {

    String error;
    String error_message;
    String amount;

    // getters/setters
```

Example of existing REST WS

Client

```
public class Client {

    public static void main(String args[]) {

        final String path = "https://www.amdoren.com/api";

        ResteasyClient client = new ResteasyClientBuilder().build();
        ResteasyWebTarget target = client.target(UriBuilder.fromPath(path));
        ServiceInterface proxy = target.proxy(ServiceInterface.class);

        Result r = proxy.convert("9xwjRjxTtnzuGKH7LcWC5Vengr52F3", "EUR", "AMD");

        System.out.println("convert: "+r.getError()+"/"+r.getError_message()
                                        +"/"+r.getAmount()) ;
    }
}
```

Interesting links

- Registry of services
  - https://www.programmableweb.com/category/all/apis
  - https://github.com/toddmotto/public-apis/blob/master/README.md
- Generation of POJO from JSON
  - https://www.site24x7.com/tools/json-to-java.html

Conclusion

- Web Services: a RPC over HTTP
- Interesting for heterogeneity as there are tools in all environment
- Recently
  - SOAP WS less used
  - Restful + XML/JSON more popular