Les 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
  - Intégration d'applications à "gros grain"
  - Unité d'intégration : le "service" (interface + contrat)
- Contraintes
  - Applications conçues indépendamment, sans avoir prévu une intégration future
  - Applications hétérogènes (modèles, plates-formes, langages)
- Conséquences
  - Pas de définition d'un modèle commun
  - Utilisation d'une base commune de niveau élémentaire
    - Pour les protocoles d'accès aux services (messages)
    - Pour la description des services (interface)
  - Outil de base : XML (car adaptable)

Les Web Services (WS)

- Apport conceptuel
  - Pas de nouveaux concepts fondamentaux ...
  - ... alors, quelle est la nouveauté ?
- Apport concret
  - Avant tout, solution au problème de l'hétérogénéité
  - Vers un schéma d'intégration et de coordination d'applications à grande échelle
  - Forte implication des principaux acteurs du domaine

Forme simple de WS : XML-RPC

Description en XML d'un appel de procédure à distance
Le type des paramètres est spécifié dans le schéma XML

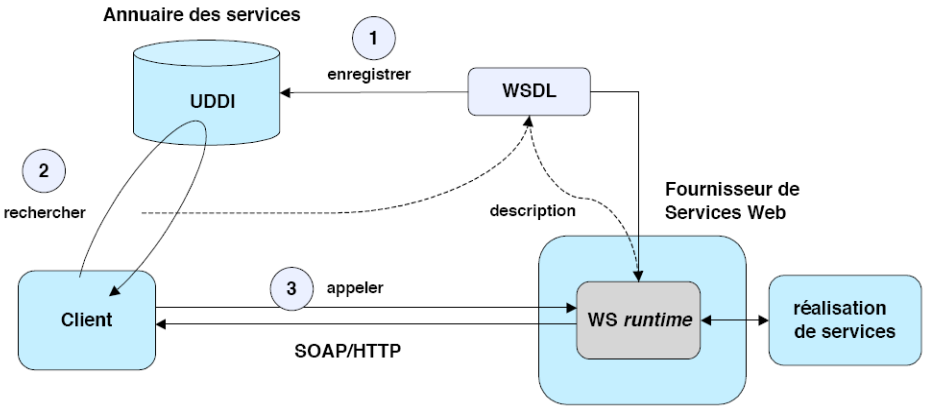
```
<methodCall>
   <methodName>meteo.temperature</methodName>
   <params>
      <param>
         <value><int>31130</int></value>
      </param>
   </params>
</methodCall>
```

Description en XML du retour des résultats

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

Intérêt : indépendance par rapport aux plates-formes et par rapport au support de communication

Mise en œuvre d'un WS



Elements des WS

- Description d'un service
  - WSDL : Web Services Description Language
  - Notation standard pour la description de l'interface d'un service
- Accès à un service
  - SOAP : Simple Object Access Protocol
  - Protocole Internet permettant la communication entre applications pour l'accès aux services Web
- Annuaire des services
  - UDDI : Universal Description, Discovery and Integration
  - Protocole pour l'enregistrement et la découverte de services

Des outils

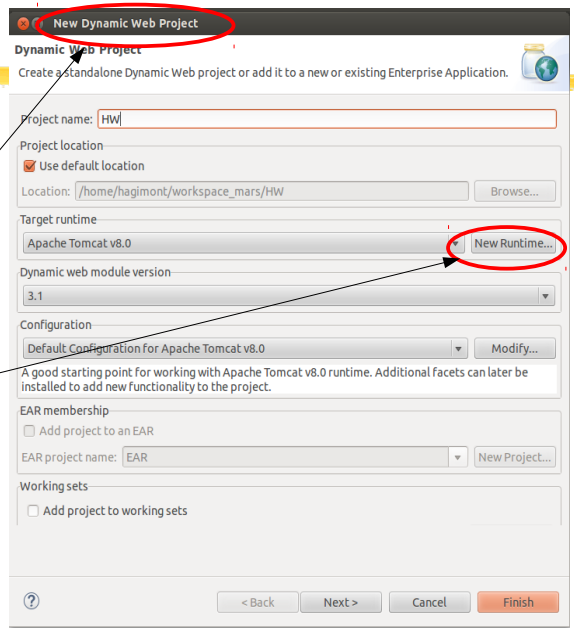
- A partir d'un programme, on générè un skeleton de WS
  - Exemple : à partir d'un programme Java, on génère
    - une servlet qui reçoit du SOAP sur HTTP et reproduit l'appel sur une instance de cette classe
    - Un fichier WDSL qui décrit l'interface du service
- Le fichier WSDL généré peut être donné aux clients
- A partir d'un fichier WSDL, on génère un stub de WS
  - Exemple : à partir d'un fichier WSDL, on génère les classes Java à utiliser pour appeler le service distant
- La programmation est simplifiée
- On dispose de tels outils dans différents environnements langages

Exemple : programmation de Web Services

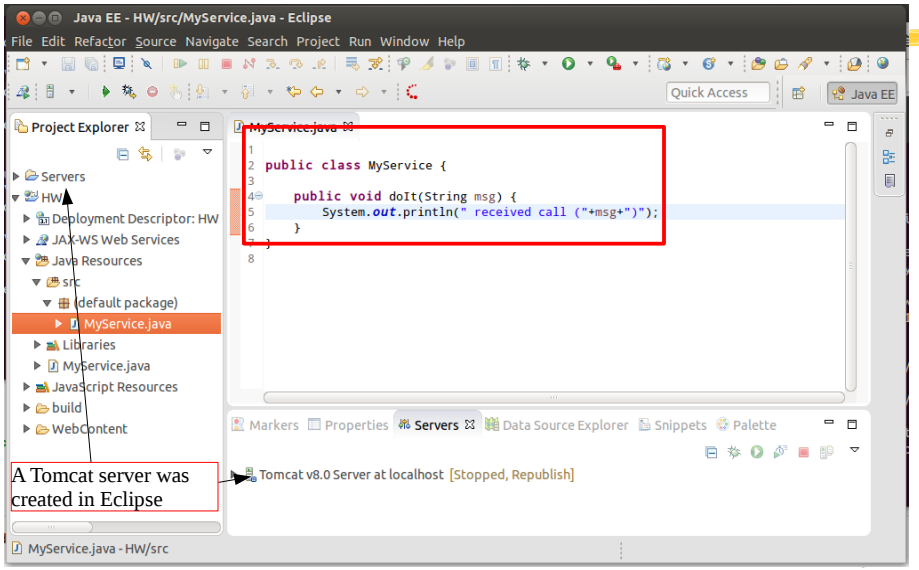
- Eclipse JEE
- Apache Axis
- Création d'un Web Service
  - A partir d'une classe Java
  - Dans le runtime Tomcat
  - Génération du fichier WSDL
- Création d'une application cliente
  - Génération des stubs à partir du fichier WSDL
  - Programmation du client

Create a Dynamic Web Project

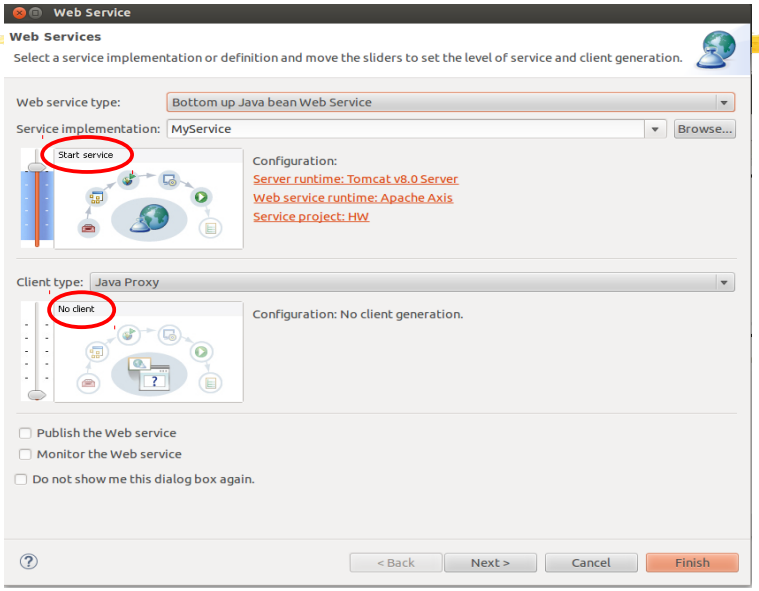
- Eclipse JEE
- Open JEE perspective
- Create a Dynamic Web Project
- Add your Tomcat runtime



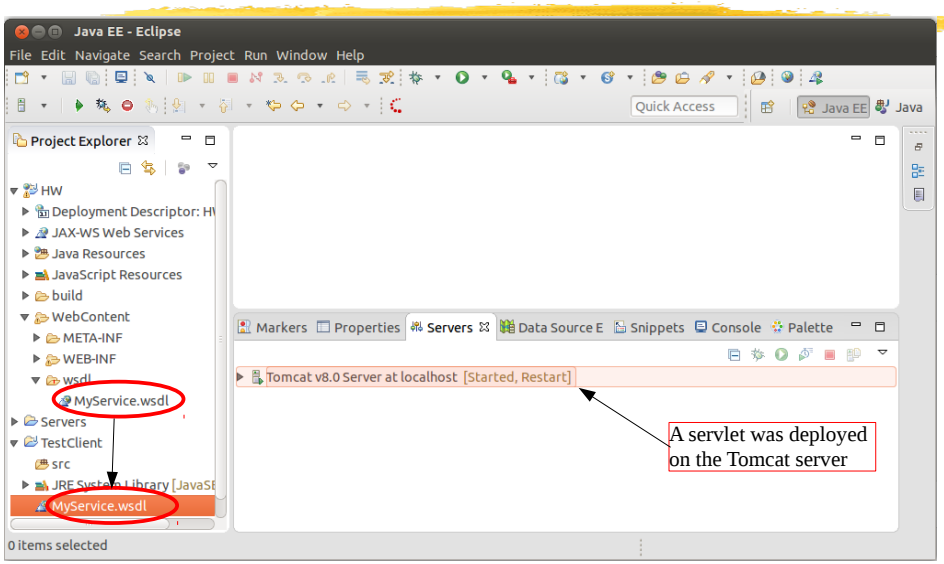
Create a Class



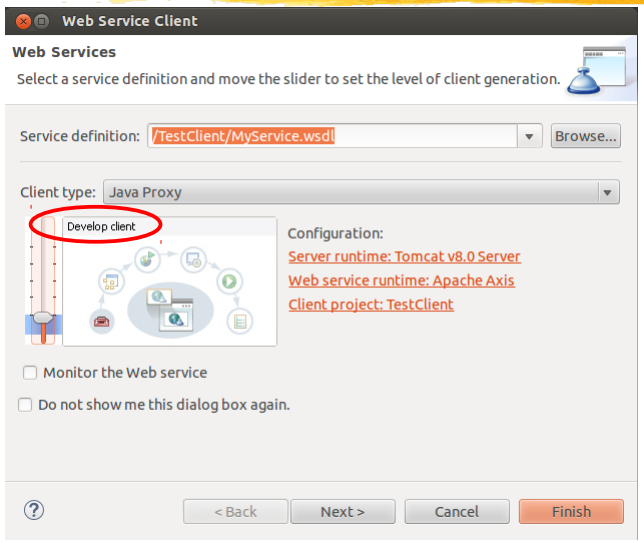
From source file : Web Service → create Web Service



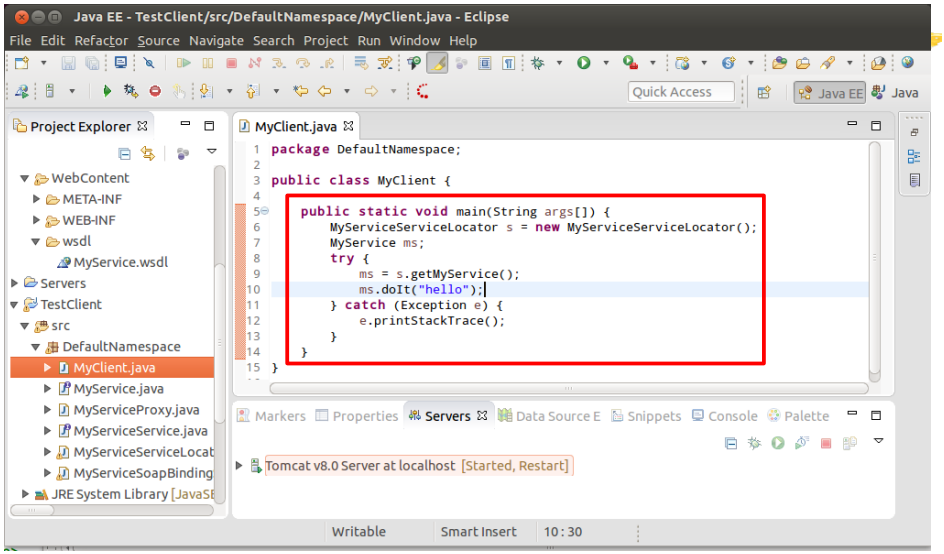
Copy the generated WSDL file in a new Java project



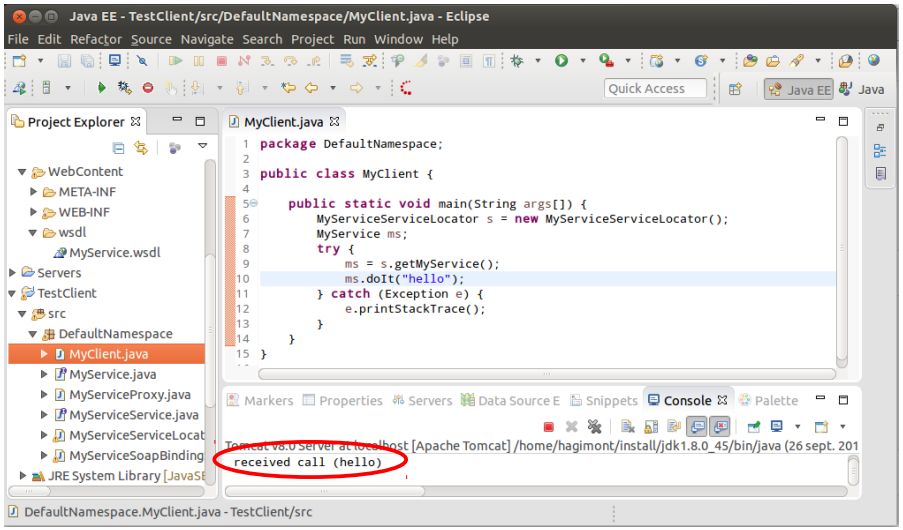
From the WSDL file
Web Service → Generate Client (Develop Client)



Program a client



Run



WSDL généré

```
<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>
```

WSDL généré

```
<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>
```


WSDL généré

```
<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>
```


Requête SOAP (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>
```


Réponse SOAP

```
<?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

- Une version simplifiée
- Appels avec les méthodes HTTP GET, POST, PUT, DELETE
- Passages de données en XML ou Json
  - Serialisation
- Spécification du protocole utilisé par le service
- Des outils d'aide au développement
  - Exemples : resteasy, jersey

Exemple avec Resteasy (serveur)

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

    // bad
    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();
    }
}
```

Person est un simple POJO

Exemple avec Resteasy (client)

```java
@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();

}
```


Exemple avec Resteasy (client)

```java
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());
   }

}
```

Exemple de service existant

- 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 }
```


Exemple de service existant

```java
@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
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()) ;
    }
}
```

Annuaire de services
https://www.programmableweb.com/category/all/apis

Conclusion

- Un RPC sur HTTP
- Mais entre langages hétérogènes
- Récemment
  - WS de moins en moins utilisé
  - Restful + XML/Json de plus en plus populaire