Web Services

Pellizzaro Massimiliano

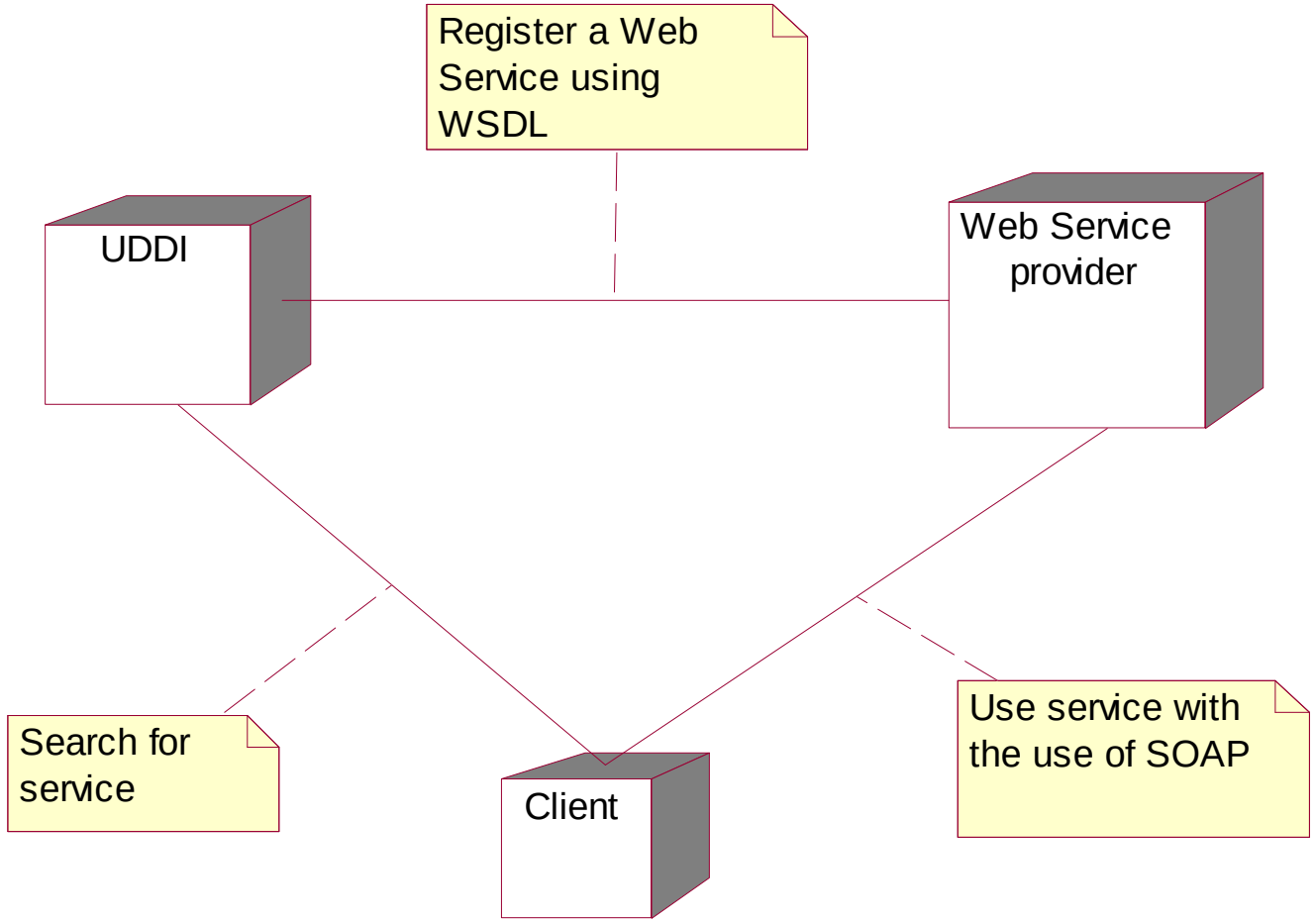
Milano, Settembre 2003

Nomenclatures

- **WSDL** : Web Services Definition Language
- **UDDI** : Universal Description Discovery and Integration
- **SOAP** : Simple Object Access Protocol

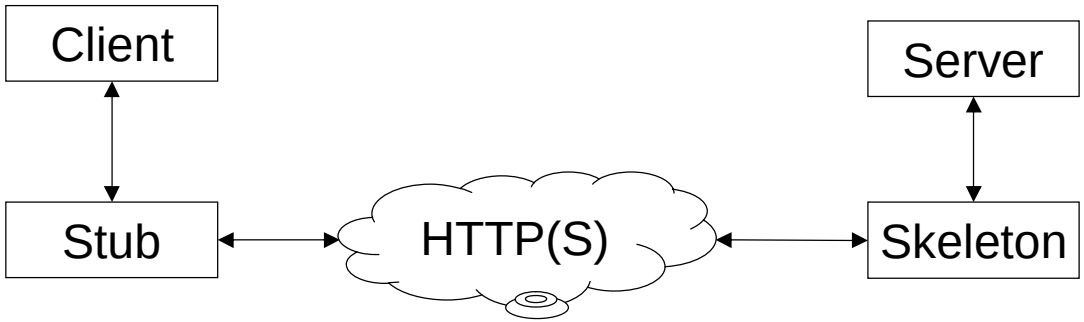
```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace=http://localhost:8081/axis/SquareNumber.jws  …….
- <wsdl:message name="SquareResponse">
  <wsdl:part name="SquareReturn" type="xsd:int" />
  </wsdl:message>
- <wsdl:message name="SquareRequest">
  <wsdl:part name="i" type="xsd:int" />
  </wsdl:message>
- <wsdl:portType name="SquareNumber">
- <wsdl:operation name="Square" parameterOrder="i">
  <wsdl:input message="intf:SquareRequest" name="SquareRequest" />
  <wsdl:output message="intf:SquareResponse" name="SquareResponse" />
  </wsdl:operation>
  </wsdl:portType>
- <wsdl:binding name="SquareNumberSoapBinding" type="intf:SquareNumber">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="Square">
  <wsdlsoap:operation soapAction="" />
```

The entire picture



Histoy of SOAP : XML-RPC

- XML-RMC is a specification for representing remote procedure calls
- It is not a standard, but it allows to perform remote procedure calls through an HTTP tunnel. This allows to pass through firewall (rmi does not allow this)



1. The client calls the remote procedure
2. The client stub marshals the call on to HTTP as a message
3. The server skeleton receives and un-marshals the message
4. The method call is executed on the server skeleton and the result is sent back to the client

XML-RPC

- XML-RPC Request
  - It is a XML document

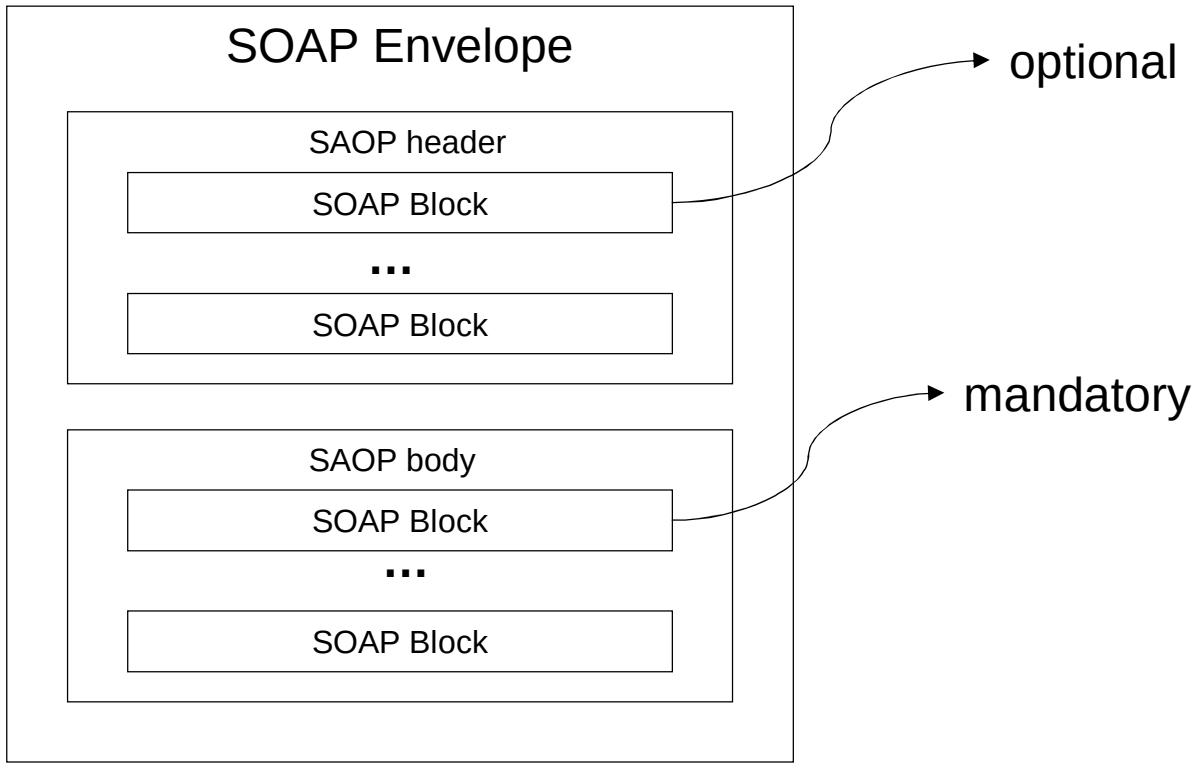
```
<methoCall>
        <methodName> getXX</metodName>
        <params>
                <param><value><int>123</int></value></param>
        </params>
</metodCall>
```

- XML-RPC Response
  - It is a XML document

```
<methoResponse>
        <value><int>123</int></value>
         <value><string>123</string></value>
</metodResponse>
```

SOAP

- **S**imple **O**bject **A**ccess **P**rotocol is a text-based protocol that uses XML for exchanging data
- Data are exchanged using a SOAP envelope



SOAP example

```
<?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>

  <ns1:Square     soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://localhost:8081/axis/SquareNumber.jws">
   <i xsi:type="xsd:int">4</i>
  </ns1:Square>

 </soapenv:Body>

</soapenv:Envelope>
```

XML-RPC vs SOAP

- There is no clean mechanism to pass XML document themselves in an XML-RPC request or response. XML document is not a type of XML-RPC, you will need to pass it as a string type.
- There is no solution that enables programmers to extend the request or response format.
- XML-RPC is not aligned with XML standardization
  - No namespaces supported
  - It defines its own data types
- XML-RPC is bound to HTTP, while SOAP 1.1 uses HTTP, SMTP and FTP transport protocol

SOAP server: Axis

- Download tomcat Servlet Container
- Downloading Axis from Apache web site
- Download an xml parser, for example Xerces
- Extract web-application called Axis and place it under tomcat webapps
- Test the installation:
  - **http://localhost:8080/axis**

Create our first Web Service

- Write a simple java class that return the square of a number

```
public class SquareNumber
{
  // Web Service method
  public int Square(int i)
  {
   return (i*i);
  }
}
```

- Rename SquareNumber.java with SquareNumber.jws
- Run http://localhost:8080/axis/SquareNumber.jws
- Run http://localhost:8080/axis/SquareNumber.jws?wsdl

Write the client code to call the Web Service

- There is a useful class that comes with Axis called :

  org.apache.axis.wsdl.Wsdl2java

- Place under the classpath the axis classes

- Run **java org.apache.axis.wsdl.WSDL2java http://localhost:8080/axis/SquareNumber.jws?wsdl**

- Four files will be generated:
  - **SquareNumber.java** -> Interface
  - SquareNumberService.java -> Interface
  - **SquareNumberServiceLocator.java** -> Class
  - SquareNumberServiceSopaBindingStub.java -> Class

Write the client code to call the Web Service

- Calling the Web Service

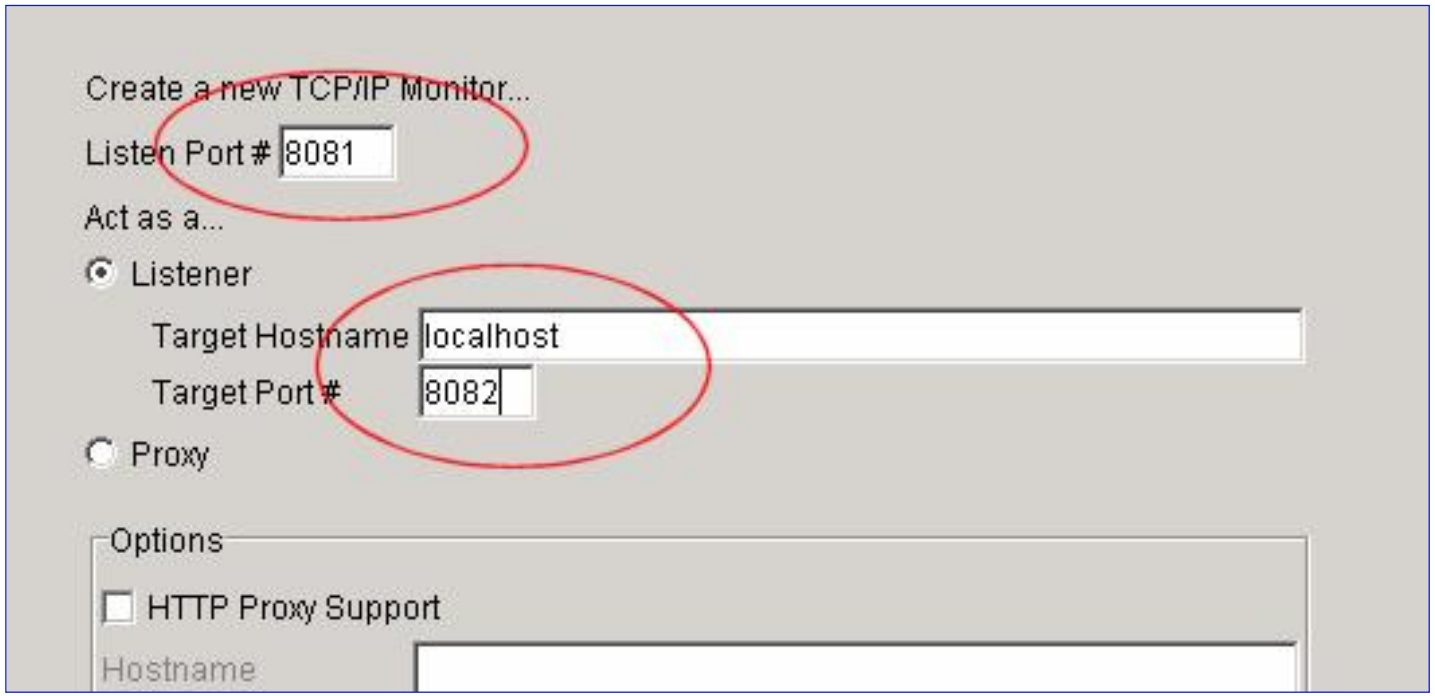
```
public class ClientCall  {
  public ClientCall(int k)   {
   try {
   SquareNumberServiceLocator serv = new SquareNumberServiceLocator();
   SquareNumber sq = serv.getSquareNumber(); // ServiceException
   int j = sq.square(k); // RemoteException
   System.out.println(j);
   }
   catch (Exception e)    {
    System.out.println("Error :" + e.toString());
   }
  }

  public static void main(String[] args)   {
    System.out.println("Sono nel main...");
    ClientCall clientCall = new ClientCall(Integer.parseInt(args[0]));
  }
}
```

- Let's try it…

Under the covers

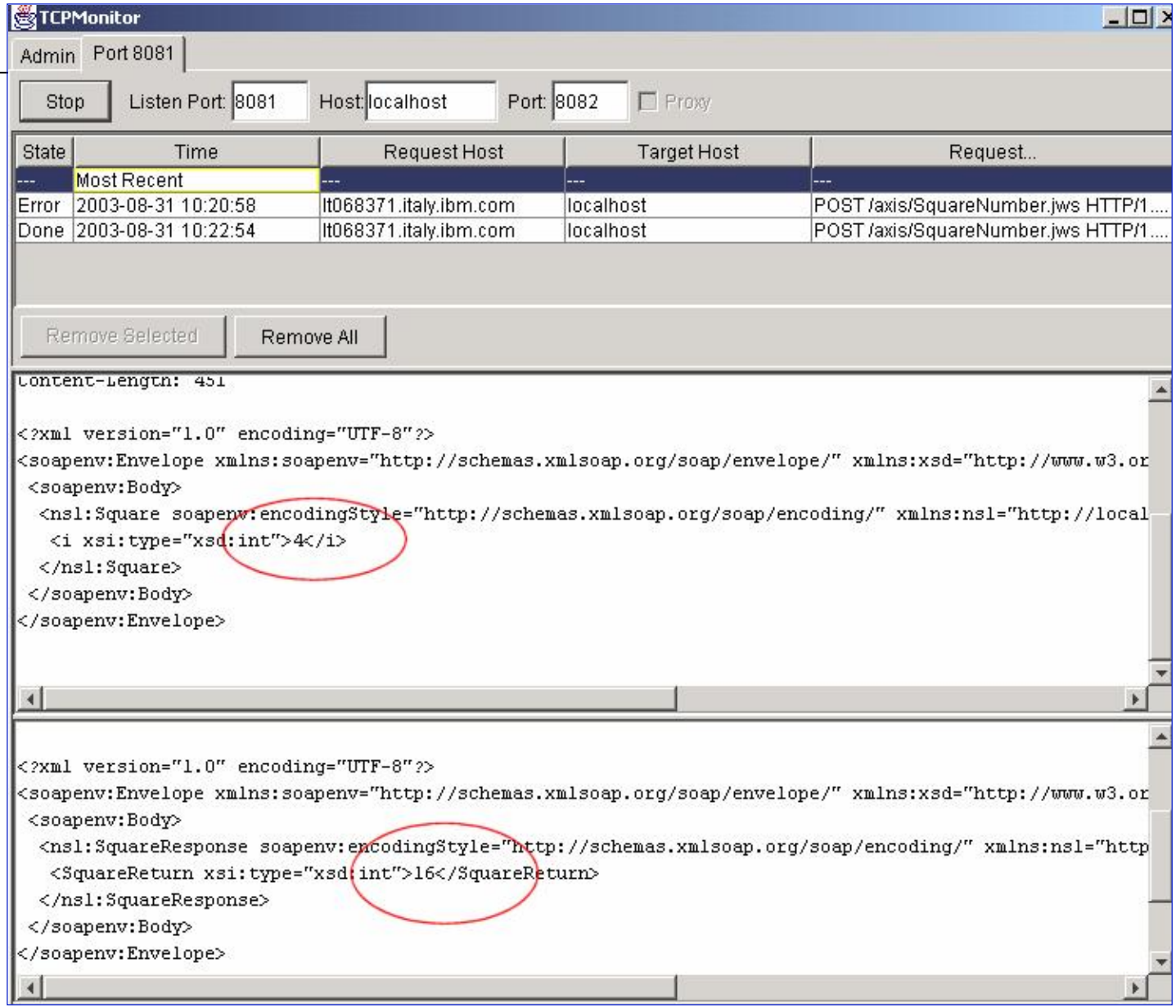
- Axis comes with a “monitor” that we can use to monitoring the messages exchanged from the client to the server
- To run the monitor use this simple command line:

  java.org.apache.axis.utils.tcpmon
- Configure the monitor to listen to the messages exchanged



The result

- Request with int 4 as a parameter
- Response with int 16 as a returned value



What we have learned

- WSDL, UDDI, SOAP, XML-RPC
- SOPA Server: Axis
- Build our first Web Service
- Build the client code to call the Web Service
- tcpMonitor

Exercise:

Build a web application that sends a parameter with an HTML form and gets the result calling a Web Service.