Web Services

Servizio Telematico Doganale

USER MANUAL

Contents

1 Introduction

A Web Service is by definition a service described by the provider in a standard and independent manner from the programming language it was developed in.
To utilize Web Services created by the provider (Servizio Telematico Doganale - STD) it is necessary to create a Web Services client.

This is possible through the WSDL (Web Services Description Language) that defines the service interface, which is the list of operations provided, the various parameters that each operation is expected to receive from those who call and the list of parameters the operation provides as an output.

A Web Service is recalled parametrically by the user of the service (requester) through application programs based on a remote call protocol, independent from the network and programming language, known as SOAP (Simple Object Access Protocol).

A WSDL is an XML document that contains a set of definitions to describe the service (Web Service). The most important elements used by WSDL are the following:

- <types>
  defines data types used in the service;
- <message>
  contains the definitions of the exchange messages of the service using parts defined as types in the “types” section;
- <portType>
  describes the service, the operations that can be performed and the messages that are involved in these operations. For each method the input message and output message are defined. The element porType can be compared to a library of functions in a traditional programming language;
- <bindings>
  contains the link between the portType (the abstract definition of the service) and the physical end-point. This information indicates which protocol to use and how to relate the input and output messages to the used protocol;
- <port>
  defines the access port of the service. A service can have multiple ports, each one with a name and a protocol binding;
- <service>
  contains the definition of the service in terms of its description and physical location (typically its URL) - defined endpoints.

The WSDL file that describes the services (Web Services) offered by the “Servizio Telematico Doganale” is available to clients, by connecting from a web browser at the URL:

test environment:

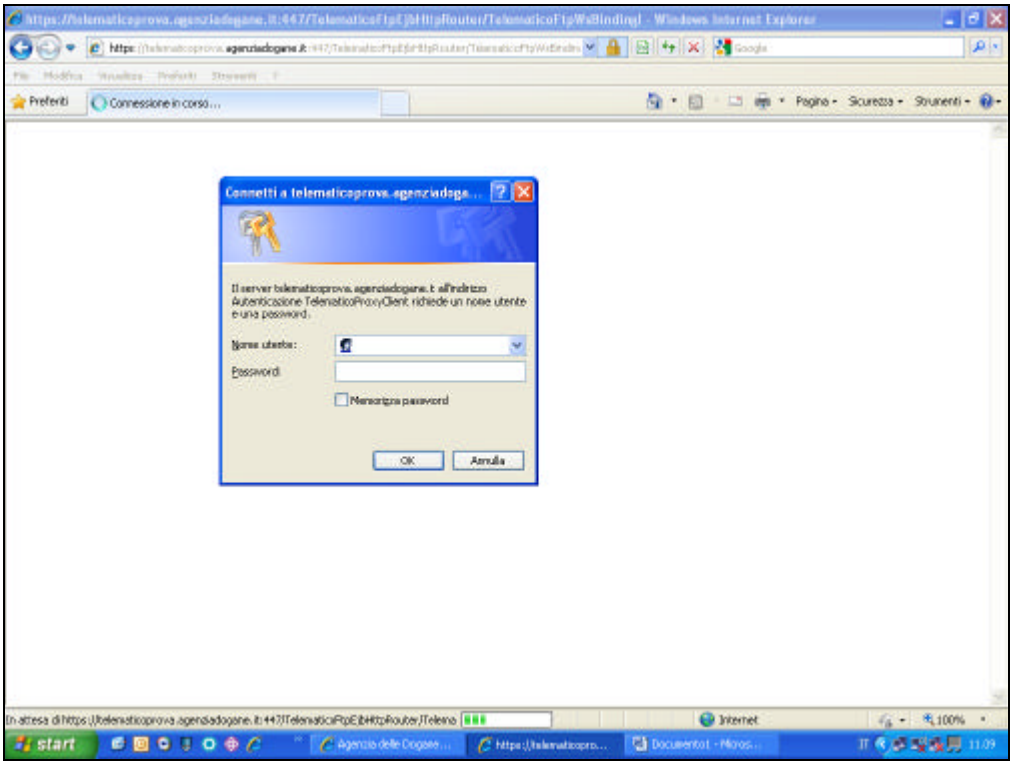
https://ws-telematicoprova.agenziadogane.it/TelematicoFtpEjbHttpRouter/TelematicoFtpWsBindingImplService/META-INF/wsdl/TelematicoFtp.wsdl

production environment:

https://ws-telematico.agenziadogane.it/TelematicoFtpEjbHttpRouter/TelematicoFtpWsBindingImplService/META-INF/wsdl/TelematicoFtp.wsdl

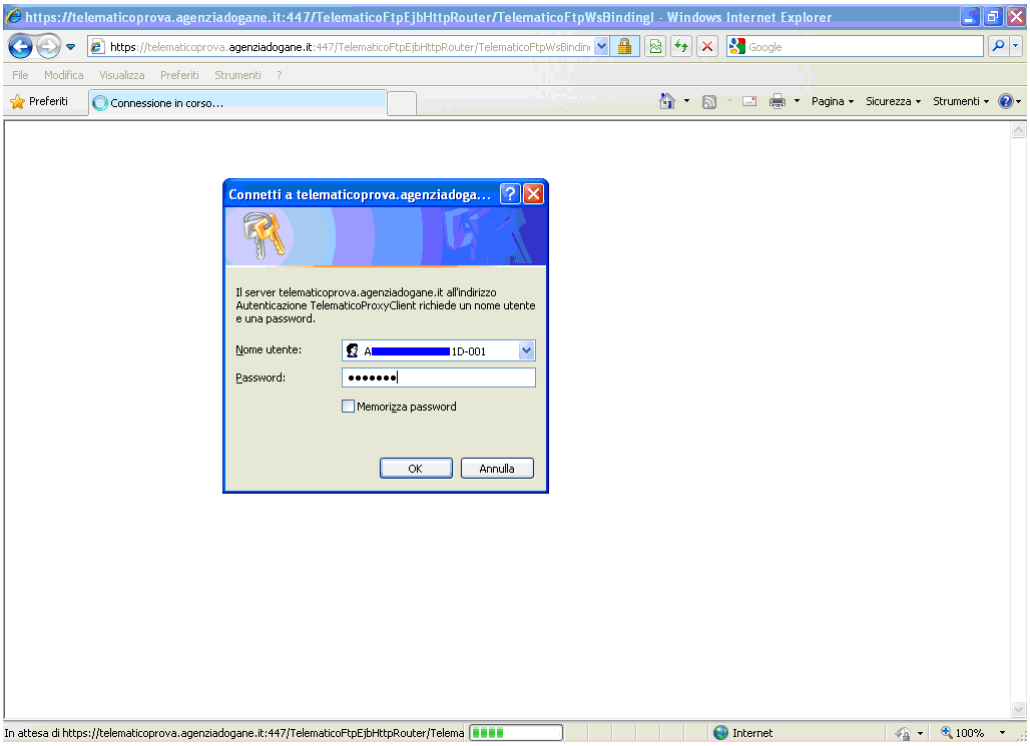
and is listed in this document in Chapter 6 - WSDL – “Servizio Telematico Doganale”.

In order to access the page that displays the WSDL file you must enter your user name and password. After typing in the URL of the WSDLthe following login page will appear:

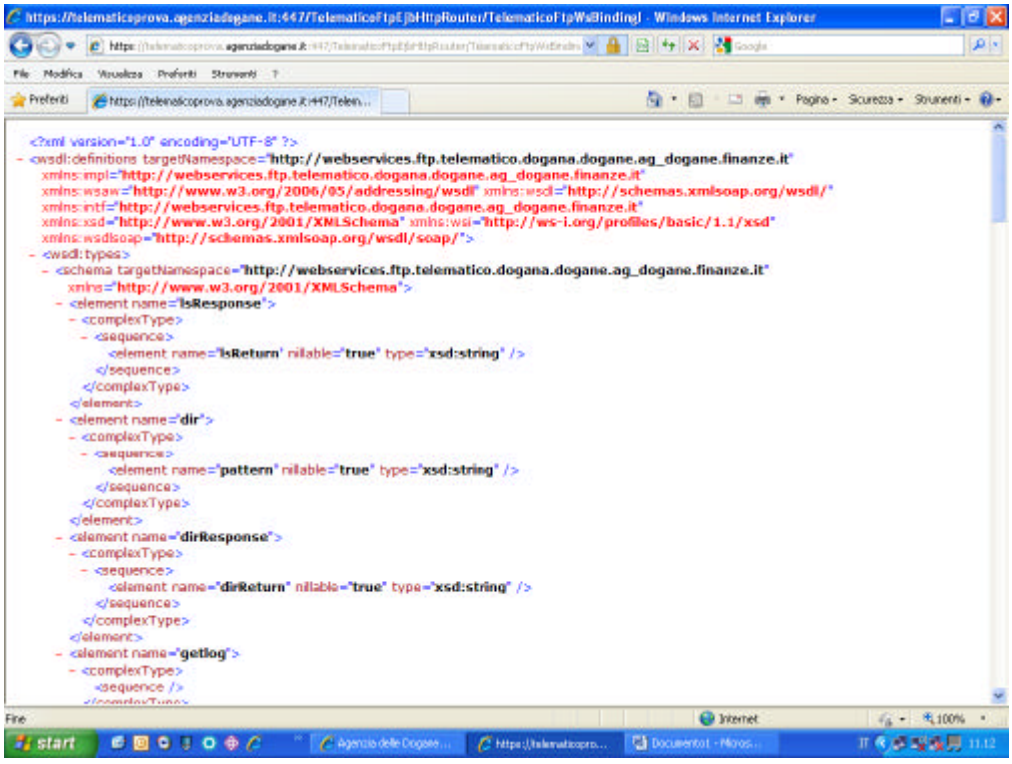


The **user name** or **nome utente** is the tax code of the subject registered to “Servizio Telmatico Doganale, followed by the symbol "-" and the progressive location (for example 0000121122-001).

The **password** is "password di accesso all’applicazione", relative to the previously indicated codice fiscale, as indicated in the print screen provided by the "Primo accesso"..



Once logged in with user name and password, the system provides the WSDL of the application;



2 Functional testing of web services

It is recommended to test the described Web Services before creating real clients.

For this purpose, the user can use, for example, the tools SoapUI, an open source Java tool, licensed under the GNU LGPL (Lesser General Public License), extremely useful and usable on any platform (Windows / Unix / Linux).

SoapUI is intended for developers and testers of web services as it allows to inspect the web services, recall them, implement them and do load tests.

The functional and load testing can be done either interactively, using a user interface, or through an automated process using a command line tool.

Regarding the functionalities of invocation and analysis of theWeb Services, SoapUI allows:

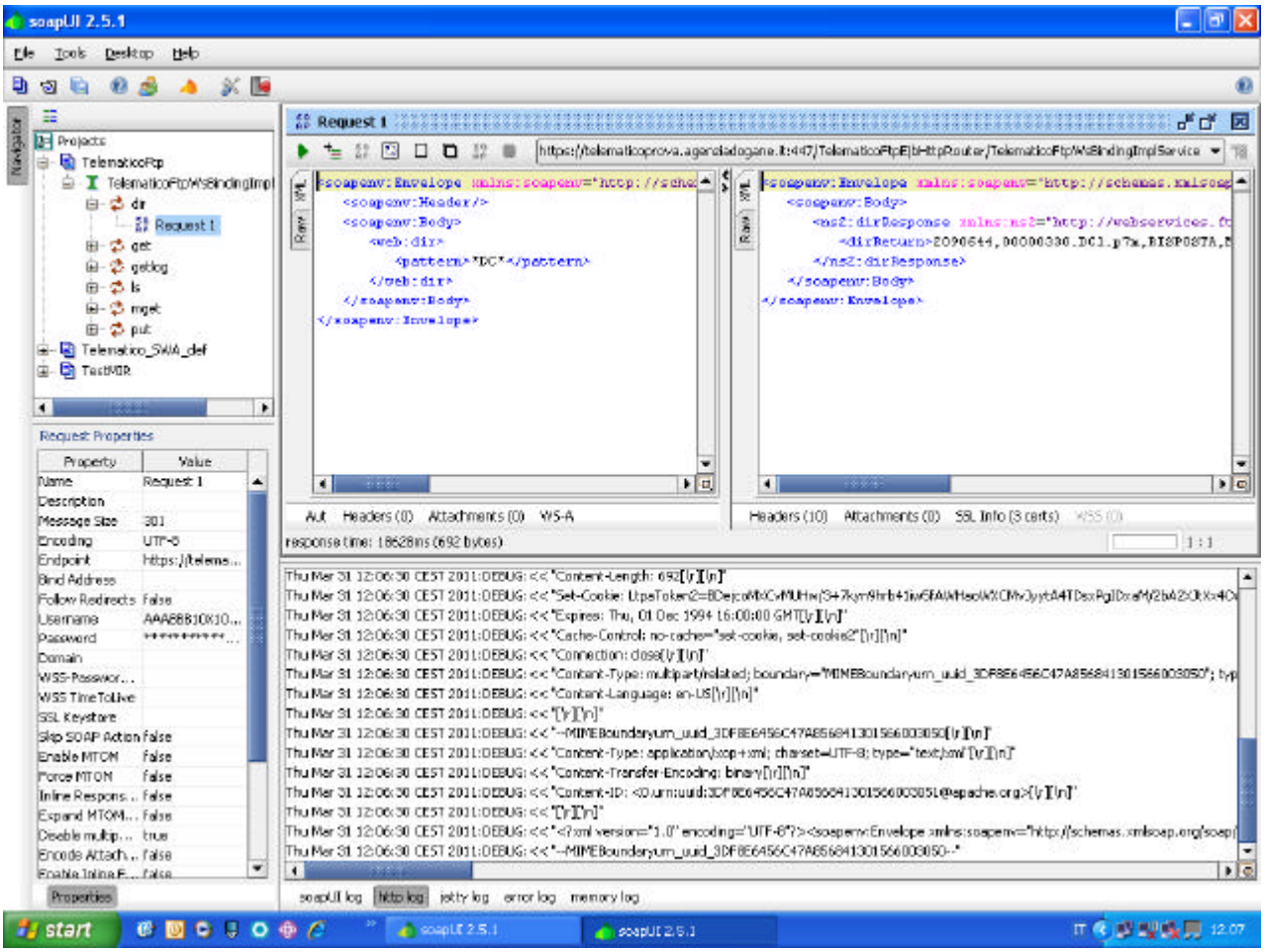
- importing the WSDL;
- automatic generation of requests;
- Support for various types of authentication (Digest, WS-Security, NTLM, etc.).
- Support for SOAP 1.1 and 1.2;
- editor with coloring syntax and function of undo / redo and automatic formatting.

Development and validation of the Web services is allowed by:

- code generation both client and server type with support for popular toolkits: JBossWS, JWSDP, Axis 1 and 2, XFire,. NET and gSOAP;
- Generation of WSDL from Java code;
- Generation of XML binding classes for JAXB and XMLBeans;
- validation of the definition of Web Services.

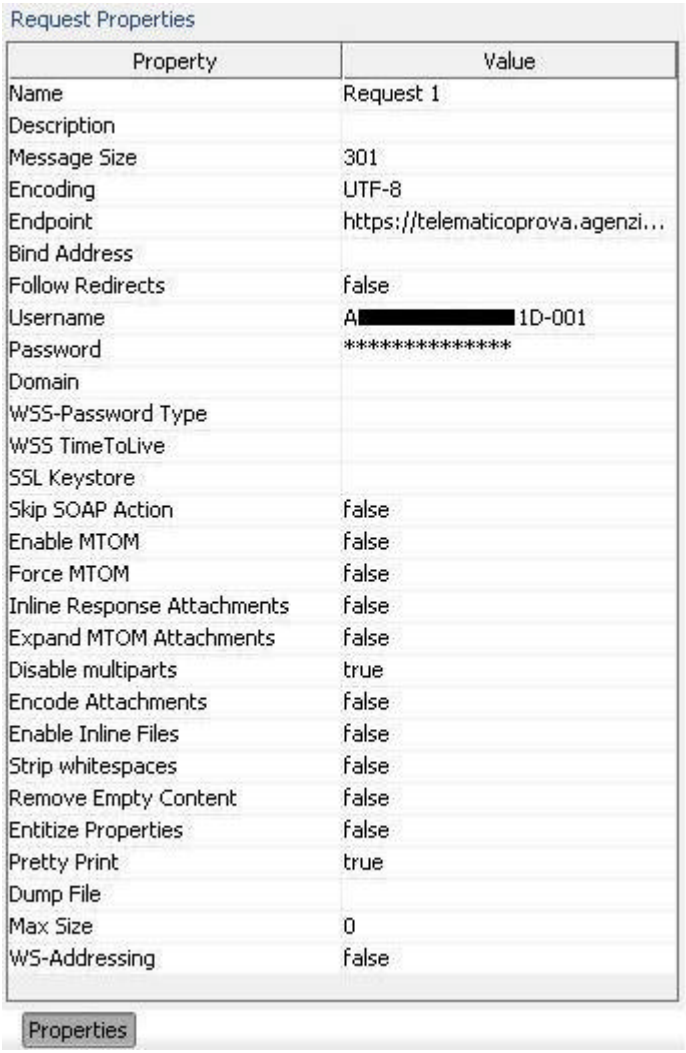
It is also possible to use powerful functions, both for functional testing and load tests of web services. In this case it is possible to have detailed reports, various statistics, logs and performance analysis.

The interface is intuitive and easy to use as shown in the figure below.



To test a web service it is sufficient to create a new project with the name and the WSDL file descriptor of the service.

Calls to the different services will be automatically created and for each call you will see a tab at the bottom where you can specify a set of properties to be added to the request.


Request Properties

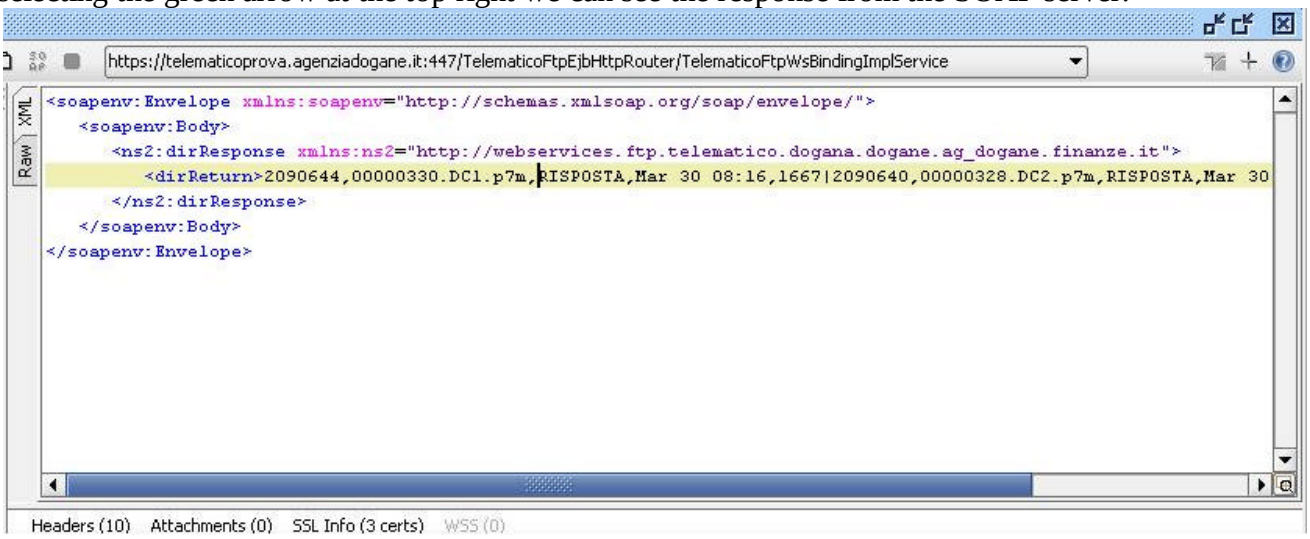
Property	Value
Name	Request 1
Description	
Message Size	301
Encoding	UTF-8
Endpoint	https://telematicoprova.agenzi...
Bind Address	
Follow Redirects	false
Username	A[redacted]1D-001
Password	**********
Domain	
WSS-Password Type	
WSS TimeToLive	
SSL Keystore	
Skip SOAP Action	false
Enable MTOM	false
Force MTOM	false
Inline Response Attachments	false
Expand MTOM Attachments	false
Disable multiparts	true
Encode Attachments	false
Enable Inline Files	false
Strip whitespaces	false
Remove Empty Content	false
Entitize Properties	false
Pretty Print	true
Dump File	
Max Size	0
WS-Addressing	false

Properties

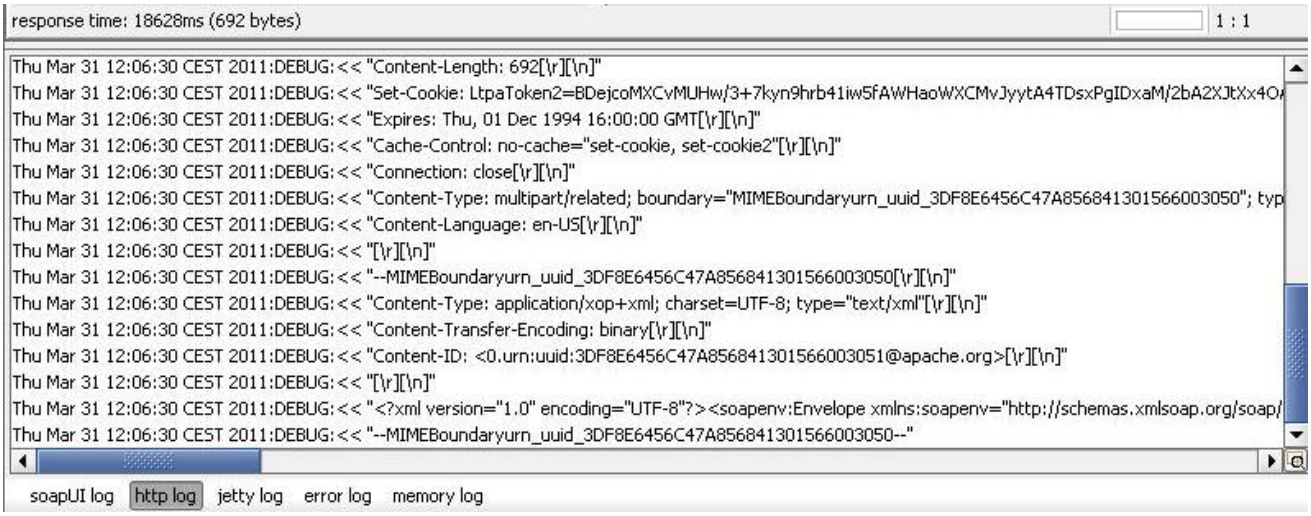
For the web services of the "Servizio Telematico Doganale" such properties let you specify the username and password to access the service.
In the middle window you can see the SOAP request. In the combo on top it is necessary to specify the service endpoint for each request:

https://ws-telematicoprova.agenziadogane.it/TelematicoFtpEjbHttpRouter/TelematicoFtpWsBindingImplService

selecting the green arrow at the top right we can see the response from the SOAP server.



Below are the various call logs.



3 Creating the client

There are several frameworks, and Axis is an example, that starting from a WSDL file create both the client and server Java code.
These frameworks can be integrated into development environments, allowing to use many features provided within graphical interfaces that help the developer.

There are several products dedicated to web services to integrate into an IDE (Integrated Development Environment), both open source and commercial.

3.1 Open Source solutions

The most popular IDE in open source are:

- ? Eclipse
- ? Netbeans

They make the top-down approach much easier because it is easy to create the web service client using a graphical interface, starting from the WSDL file.

We have to remember that a Web service client may itself be a web application development environment and then in that case would need an application server support.

Eclipse is an integrated cross-platform and multi-language development environment. It is developed by a community structured in an open source style. One of the most useful components for the development of Web applicationsis certainly the one belonging to the project Web Tools Platform (WTP). WTP is a platform that extends the base functionality of Eclipse and includes various tools that can substantially facilitate the work of the developer of web services; among the features that stand out are a specific editor, tools for testing and API to support the programmer during the coding. The application server that is usually used with Eclipse is Tomcat.

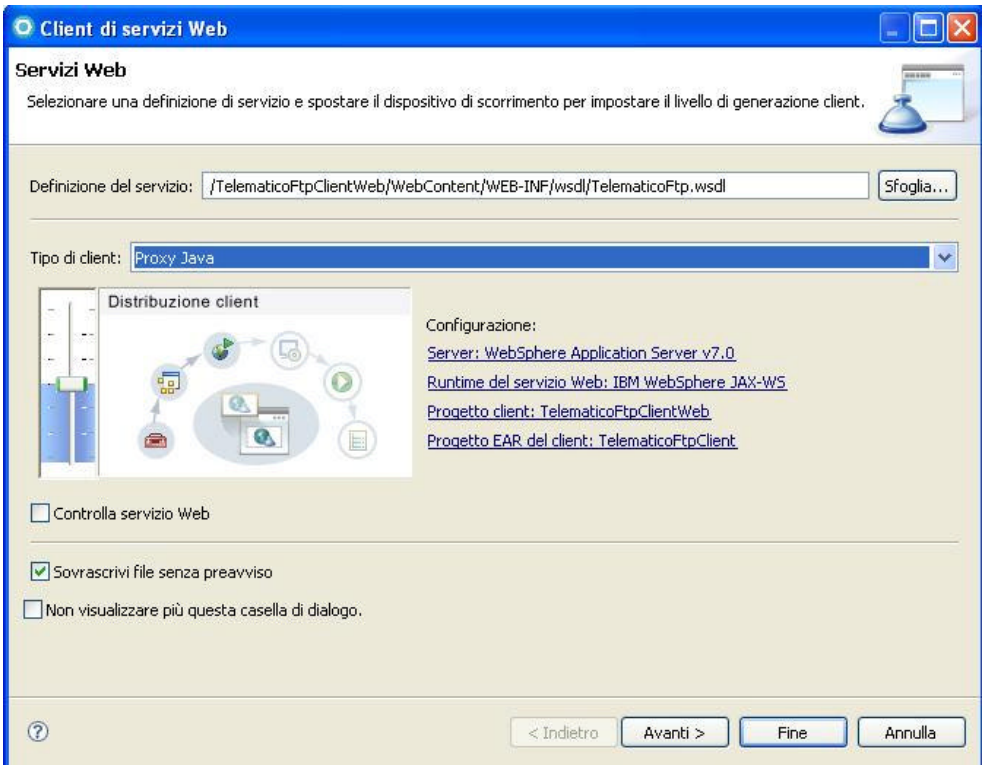
NetBeans is a multi-language development environment written entirely in Java. It was chosen by Sun Microsystems as the official IDE. It has many interesting plug-ins and wizards for developing Web Services easily. Glassfish is the application server normally used, undoubtedly an excellent solution.

3.2 Proprietary Solution

In addition to open source solutions described above, there are several proprietary solutions for the creation of a Web service client including, for example:

- ? RAD (Rational® Application Developer) IBM.
- ? Microsoft Visual Studio®

In RAD the creation of Java classes, useful to the generation of the client, is easy and intuitive. Just enter the WSDL file in the project, then right click select "Web Services" and then "Generate Client", the following window will appear:



Here the developer can select various configuration options including, for example, the runtime of the service (JAX-WS in our example).

Visual Studio is an integrated development environment by Microsoft, which currently supports different types of language. Visual Studio allows to create client web services through the integration of. NET framework. Given the WSDL of the Web service, we can create a class as a proxy for a page in ASP.NET, which is a component that allows us to transparently access the Web Service, without worrying about the manner in which this is called and used. The proxy class will then provide the service exactly as if it were a component made by us and used in the project.

The command-line utility wsdl.exe SDK. NET allows us to generate a proxy class:

wsdl / language: CS http://localhost/webservice/wsftp.asmx?WSDL

The language parameter allows us to specify one of the languages supported by. NET framework (in this example is required to wsdl.exe to generate a C # class).
The result of this transaction is the generation of a wsftp.cs file containing the proxy class code for the service.
At this point we just have to fill in the above file, again using the SDK:

csc / t: library / out: wsftp.dll wsftp.cs / r: system.dll / r: System.Xml.dll / r: system.web.services.dll

We have compiled the class as a class library (/ t: library) indicating the result of compiling and using wsftp.dll metadata specified in the parameters / r.

At this point the file wsftp.dll must be entered in the / BIN directory of the application that uses it.

4 Basic Features

4.1 LS

This feature allows you to extract and deliver the list of response and outcome files for the authorized user of the service by searching on the basis of a specific pattern and within a given time interval.

- **Input**: represents the pattern according to which the request is filtered
- **Output**: the string that contains the outcome of the operation and the query result

Here is an example of the output, appropriately revised in the client for the input pattern: **00001209.***

00001209.B99
00001209.UYY.p7m
00001209.UXX.p7m
00001209.UCC.p7m
00001209.UBB.p7m
00001209.UAA.p7m
00001209.UB2.p7m
00001209.UB1.p7m
00001209.P99

4.2 DIR

The function provides a list of response and outcome files along with the date of the deployment of the "Servizio Telematico Doganale".
The function searches in a specified time interval.

- **Input**: Pattern to filter the request
- **Output**: A string containing the result of

Here is an example of an output, appropriately revised by a possible client application with the input pattern: **00001209.***

14076171,00001209.B99,RISPOSTA,Dec 28 19:18,112
14076157,00001209.UYY.p7m,RISPOSTA,Dec 09 17:05,1890
14076156,00001209.UXX.p7m,RISPOSTA,Dec 09 16:42,1890
14076155,00001209.UCC.p7m,RISPOSTA,Dec 09 16:13,1890
14076154,00001209.UBB.p7m,RISPOSTA,Dec 09 16:12,1890
14076153,00001209.UAA.p7m,RISPOSTA,Dec 09 13:32,1890
14076152,00001209.UB2.p7m,RISPOSTA,Dec 09 13:19,1890
14076171,00001209.P99,ESITO,Dec 09 19:22,330

The figures reported are in the order: code files, file name received, receipt type, date, size received.

4.3 PUT

The function realizes the transmission of a file to the “Servizio Telematico Doganale”..

? **Input**: a string containing the file name and an array of bytes for the file itself

? **Output:** A string containing the result of the operation.
If successful, the output returns the file name, date of transmission and the code of the file.
In case of a negative output, a report with the error code is produced.

It should be noted that since the function takes as an input the representation shown in bytes of the file you want to transmit to the “Servizio Telematico Doganale”., the initial reading of the file and the following transformation into arrays of bytes are to be entrusted to the client application.

? Here's an example of a successful outcome:

Upload del file**:* *C:\Programmi\FirmaVerifica2.1\firmati**00001122.I08.p7m

***Output** :* *Caricato con successo.*

Here's an example of the output to fail, because the user logged on user code has enabled 0000 and sent a file whose name has the first 4 characters other than this code:

Upload file**:* *C:\Programmi\FirmaVerifica2.1\firmati**057R1209.I0B.p7m

Output : *Nome file fuori formato: deve iniziare con le prime 4 lettere del codice utente.*

4.4 GET

This function allows the download of a response or outcome file as requested by the user.

? **Input**: A string containing the name of the file to download.

? **Output**: The required file in byte array format.

4.5 GETLOG

This feature allows to download the error log file operations in a given time interval.

Example of the contents of the log file:
[06/12/2010 12:26:20] 000010.13.CD2.p7m : Nome file fuori formato: la lunghezza è errata
[06/12/2010 09:48:13] 00000126.HAA : Il file non appartiene ad una tipologia abilitata
[06/12/2010 13:49:27] 057R1209.I0B.p7m : Nome file fuori formato: deve iniziare con le prime 4 lettere del codice utente
[07/12/2010 15:33:26] 00001122.I05 : Verificare che il file sia stato firmato
[07/12/2010 15:35:52] 0103.Ra11 : Nome file fuori formato: la lunghezza è errata

5 Security

Access to Web Services is regulated by BASIC type authentication for the passage of credentials to be verified by the currently existing LDAP system.

During the construction of the client application it will therefore be necessary to define, according to the jaxws standards, the access policiesfor the Web service.

The user must specify that username and password when calling the different services.

Below is an example of a call made from the transaction DIR java:

```
TelematicoFtpWsBindingImplProxy service = new TelematicoFtpWsBindingImplProxy();
BindingProvider bp = (BindingProvider) service._getDescriptor().getProxy();
bp.getRequestContext().put(BindingProvider.USERNAME_PROPERTY, "USERNAME");
bp.getRequestContext().put(BindingProvider.PASSWORD_PROPERTY, "PASSWORD");
String result = service.dir(pattern);
```

6 WSDL – Servizio Telematico Doganale

```
<?xml version="1.0" encoding="UTF-8" ?>
-<wsdl:definitions
    targetNamespace="http://webservices.ftp.telematico.dogana.dogane.ag_dogane.finanze.it"
    xmlns:impl="http://webservices.ftp.telematico.dogana.dogane.ag_dogane.finanze.it"
    xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:intf="http://webservices.ftp.telematico.dogana.dogane.ag_dogane.finanze.it"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsi="http://ws-
    i.org/profiles/basic/1.1/xsd" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/">
-<wsdl:types>
-<schema
    targetNamespace="http://webservices.ftp.telematico.dogana.dogane.ag_dogane.finanze.it"
    xmlns="http://www.w3.org/2001/XMLSchema">
-<element name="lsResponse">
-<complexType>
-<sequence>
 <element name="lsReturn" nillable="true" type="xsd:string" />
    </sequence>
    </complexType>
    </element>
-<element name="dir">
-<complexType>
-<sequence>
 <element name="pattern" nillable="true" type="xsd:string" />
    </sequence>
    </complexType>
    </element>
-<element name="dirResponse">
-<complexType>
-<sequence>
 <element name="dirReturn" nillable="true" type="xsd:string" />
    </sequence>
    </complexType>
    </element>
-<element name="getLog">
-<complexType>
 <sequence />
    </complexType>
    </element>
-<element name="getLogResponse">
-<complexType>
-<sequence>
 <element name="getLogReturn" type="xsd:base64Binary" />
    </sequence>
    </complexType>
```

```
    </element>
-<element name="get">
-<complexType>
-<sequence>
 <element name="fileName" nillable="true" type="xsd:string" />
    </sequence>
    </complexType>
    </element>
-<element name="getResponse">
-<complexType>
-<sequence>
 <element name="getReturn" type="xsd:base64Binary" />
    </sequence>
    </complexType>
    </element>
-<element name="put">
-<complexType>
-<sequence>
 <element name="fileMsg_Content" type="xsd:base64Binary" />
 <element name="fileMsg_Name" nillable="true" type="xsd:string" />
    </sequence>
    </complexType>
    </element>
-<element name="putResponse">
-<complexType>
-<sequence>
 <element name="putReturn" nillable="true" type="xsd:string" />
    </sequence>
    </complexType>
    </element>
-<element name="mget">
-<complexType>
-<sequence>
 <element name="mGetInput_Name" nillable="true" type="xsd:string" />
 <element name="mGetInput_Prog" nillable="true" type="xsd:string" />
    </sequence>
    </complexType>
    </element>
-<element name="mgetResponse">
-<complexType>
-<sequence>
 <element name="mgetReturn" type="xsd:base64Binary" />
    </sequence>
    </complexType>
    </element>
-<element name="ls">
-<complexType>
-<sequence>
```

```
<element name="pattern" nillable="true" type="xsd:string" />
    </sequence>
    </complexType>
    </element>
    </schema>
    </wsdl:types>
-<wsdl:message name="putRequest">
 <wsdl:part name="parameters" element="impl:put" />
    </wsdl:message>
-<wsdl:message name="putResponse">
 <wsdl:part name="parameters" element="impl:putResponse" />
    </wsdl:message>
-<wsdl:message name="getlogRequest">
 <wsdl:part name="parameters" element="impl:getlog" />
    </wsdl:message>
-<wsdl:message name="dirRequest">
 <wsdl:part name="parameters" element="impl:dir" />
    </wsdl:message>
-<wsdl:message name="mgetRequest">
 <wsdl:part name="parameters" element="impl:mget" />
    </wsdl:message>
-<wsdl:message name="getResponse">
 <wsdl:part name="parameters" element="impl:getResponse" />
    </wsdl:message>
-<wsdl:message name="lsRequest">
 <wsdl:part name="parameters" element="impl:ls" />
    </wsdl:message>
-<wsdl:message name="lsResponse">
 <wsdl:part name="parameters" element="impl:lsResponse" />
    </wsdl:message>
-<wsdl:message name="getRequest">
 <wsdl:part name="parameters" element="impl:get" />
    </wsdl:message>
-<wsdl:message name="mgetResponse">
 <wsdl:part name="parameters" element="impl:mgetResponse" />
    </wsdl:message>
-<wsdl:message name="dirResponse">
 <wsdl:part name="parameters" element="impl:dirResponse" />
    </wsdl:message>
-<wsdl:message name="getlogResponse">
 <wsdl:part name="parameters" element="impl:getlogResponse" />
    </wsdl:message>
-<wsdl:portType name="TelematicoFtpWsBindingImpl">
-<wsdl:operation name="ls">
 <wsdl:input name="lsRequest" message="impl:lsRequest" />
 <wsdl:output name="lsResponse" message="impl:lsResponse" />
    </wsdl:operation>
-<wsdl:operation name="dir">
```

```
<wsdl:input name="dirRequest" message="impl:dirRequest" />
<wsdl:output name="dirResponse" message="impl:dirResponse" />
</wsdl:operation>
-<wsdl:operation name="getLog">
<wsdl:input name="getLogRequest" message="impl:getLogRequest" />
<wsdl:output name="getLogResponse" message="impl:getLogResponse" />
</wsdl:operation>
-<wsdl:operation name="get">
<wsdl:input name="getRequest" message="impl:getRequest" />
<wsdl:output name="getResponse" message="impl:getResponse" />
</wsdl:operation>
-<wsdl:operation name="put">
<wsdl:input name="putRequest" message="impl:putRequest" />
<wsdl:output name="putResponse" message="impl:putResponse" />
</wsdl:operation>
-<wsdl:operation name="mget">
<wsdl:input name="mgetRequest" message="impl:mgetRequest" />
<wsdl:output name="mgetResponse" message="impl:mgetResponse" />
</wsdl:operation>
</wsdl:portType>
-<wsdl:binding name="TelematicoFtpWsBindingImplSoapBinding"
    type="impl:TelematicoFtpWsBindingImpl">
<wsaw:UsingAddressing wsdl:required="false"
    xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl" />
<wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
-<wsdl:operation name="ls">
<wsdlsoap:operation soapAction="ls" />
-<wsdl:input name="lsRequest">
<wsdlsoap:body use="literal" />
</wsdl:input>
-<wsdl:output name="lsResponse">
<wsdlsoap:body use="literal" />
</wsdl:output>
</wsdl:operation>
-<wsdl:operation name="dir">
<wsdlsoap:operation soapAction="dir" />
-<wsdl:input name="dirRequest">
<wsdlsoap:body use="literal" />
</wsdl:input>
-<wsdl:output name="dirResponse">
<wsdlsoap:body use="literal" />
</wsdl:output>
</wsdl:operation>
-<wsdl:operation name="getLog">
<wsdlsoap:operation soapAction="getLog" />
-<wsdl:input name="getLogRequest">
<wsdlsoap:body use="literal" />
</wsdl:input>
```

```
-<wsdl:output name="getbgResponse">
 <wsdlsoap:body use="literal" />
   </wsdl:output>
   </wsdl:operation>
-<wsdl:operation name="get">
 <wsdlsoap:operation soapAction="get" />
-<wsdl:input name="getRequest">
 <wsdlsoap:body use="literal" />
   </wsdl:input>
-<wsdl:output name="getResponse">
 <wsdlsoap:body use="literal" />
   </wsdl:output>
   </wsdl:operation>
-<wsdl:operation name="put">
 <wsdlsoap:operation soapAction="put" />
-<wsdl:input name="putRequest">
 <wsdlsoap:body use="literal" />
   </wsdl:input>
-<wsdl:output name="putResponse">
 <wsdlsoap:body use="literal" />
   </wsdl:output>
   </wsdl:operation>
-<wsdl:operation name="mget">
 <wsdlsoap:operation soapAction="mget" />
-<wsdl:input name="mgetRequest">
 <wsdlsoap:body use="literal" />
   </wsdl:input>
-<wsdl:output name="mgetResponse">
 <wsdlsoap:body use="literal" />
   </wsdl:output>
   </wsdl:operation>
   </wsdl:binding>
-<wsdl:service name="TelematicoFtpWsBindingImplService">
 <wsdl:port name="TelematicoFtpWsBindingImpl"
     binding="impl:TelematicoFtpWsBindingImplSoapBinding" />
   </wsdl:service>
   </wsdl:definitions>
```