XML-RPC vs. SOAP vs. REST web services in Java – uniform using WSWrapper

Adina Ploscar

***Abstract*—** Every day more businesses migrate towards the web. They offer on-line support for their clients. This support translated in web pages must be linked with the companies'data servers. The solution for this are web services. With the advent of various web services also appear the problem of cross-library and cross-language compatibility. The solution to this could be a wrapper which hide the implementation details of various distributions. We call this wraper WSWrapper. The scope of WSWrapper is to provide an unique interface by wrapping some existing and widely used libraries for all XML-RPC, SOAP and REST web service. WSWrapper offers solutions in four languages most used at the moment : java, php, c#, and python. In this paper, we will refer to WSWrapper from Java view. We will see the unique interface of WSWrapper for all three models and examples of a web service and a web service client.

***Keywords*—**Java, paradigms, web service, wrapper.

I. Introduction

Currently the web tends to take up increasingly more important and higher place in daily life of all people. Its extensive development has made its contents to be extremely varied finding a place for all web application developers. This existing amalgam tends to remodel. Viewed as a whole the web wants to be formed from a variety of service providers and a lot of customers for services. This virtual life created quite chaotic lately tends to resemble more with real life where on the one hand we have the manufacturers and on the other hand we have the consumers which are all very different. As Semantic Web presented in [6] is designed to be meaningful to computers as well as to humans we need something like this for web services to the programmers too.

Over time in this "virtual life" appeared different models of web services and clients for them. The most commonly used models these days are XML-RPC, SOAP, respectively REST.

For each of these models various developers have implemented software for the most popular languages. The result was a lot of software approaches more or less different that ultimately give the same result in a shorter or longer time, with a higher or lower consumption of resources. Thus a programmer who wants to implement a web service in one of the above models will need a lot of time to study implementations offered by different developers for the three different models. And after a lot of study finally he would be able to choose the best solution for the problem he has to do.

A. P. Author is with Faculty of Natural Sciences, Engineering and Computer Science, "Vasile Goldis" Western University, Arad 310025, Bd. Revolutiei nr. 94-96, ROMANIA (corresponding author to provide phone: +40745299972; e-mail: adina_ploscar@yahoo.com)

Our open source project WSWrapper originally appeared from the idea of bringing together under one format (a single interface) all three models of Web services. Thus a programmer is no longer forced to learn one tutorial for each type of service. However he has now the posibility through this unique interface to select the type of service desired, and desired implementation language and the open source package used to accomplish the service. As mentioned in [3] WSWrapper will be available in C #, Java, PHP and Python for all types of web services: XML-RPC, SOAP, REST [9].

In the following sections is presented the model of this wrapper resulted from the extrapolation of the three types of services. Each of the three models has its own features which must be respected. WSWrapper being a common interface to all three types of services will meet the requirements of each model.

II. XML-RPC Model

The operation mode of an XML-RPC web service is as follows: any XML-RPC call is composed of two parts namely client (the caller part) and server (the called part). The server is available at a certain URL, the client will use that URL to locate the server and call a particular method, as follows:

1. The client calls a procedure using XML-RPC client, specifying a method, its parameters and a server where is that method.
2. The XML-RPC client packs the method name and parameters in an XML document and then sends it as a HTTP POST request to the server.
3. An HTTP server on the server machine receives the POST request and passes the XML content to a XML-RPC listener.
4. The XML-RPC listener reads the XML document to get the method name and parameters and then call the method with appropriate parameters.
5. The method returns a response to the XML-RPC listener which converts the response into an XML document and passes it to the web server.
6. The Web server returns the XML document as the response to the initial HTTP POST request.
7. The XML-RPC client analyzes the XML document and get the value returned by the method and passes it to the client

The client continues its execution with the return value obtained.

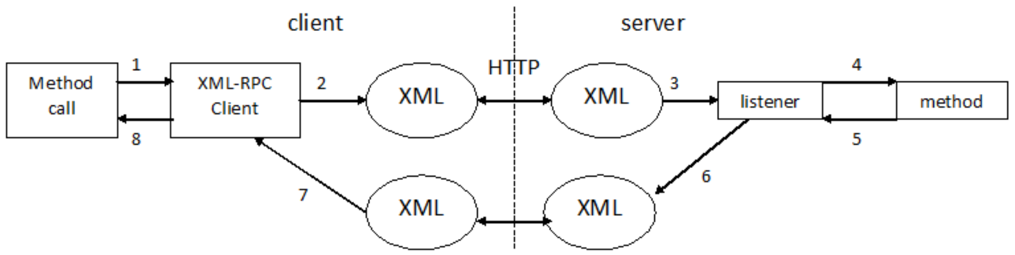


Fig. 1 The operation mode of XML-RPC

Like transport protocol XML-RPC requires the use of HTTP. Imposing HTTP as the transport protocol involve fairly significant drawbacks in terms of security and also means that XML-RPC requests are stateless and synchronous.

From the Fig.1 it is seen that the call is synchronous, the client remains on hold until the reply. If the response time is quite small, this synchronization does not disturb the processing. But if the response time is relatively large, then the programmers can build their own system of asynchronism using the tools provided by the platform on which they work.

Also being used HTTP as transport protocol, calls will be stateless. Two successive calls from the same client to the server are viewed as totally independent requests. Again if it is to keep the state, the programmers are required to develop their own tools for this. They can do this by storing a series of information in a file on the server or by using session objects or cookies to maintain session state if the server platform allows.

III. SOAP Model

The SOAP-based web services are more complex and extensive than XML-RPC based services. In terms of distributed computing paradigm, SOAP is comparable to that offered by paradigms like RMI, CORBA, EJB and DCOM.

SOAP originally came from Simple Object Access Protocol, but now can also mean Service Oriented Architecture (SOA) protocol. However, SOAP is no longer an acronym, is an XML dialect is which the messages are written. So SOAP fits in SOA paradigm like RMI, CORBA, EJB and DCOM.

The SOA concept is not new. The difference between SOA and other architectures is the missed connection meaning that the customer service is independent of the service. Some of the top reasons for using SOA concept are: reusage, interoperability, scalability, flexibility and cost effectiveness [7,8,11].

In SOA paradigm we have three actors, namely service name (which will link clients and server), server (one that provides services) and client (one that wants services). Once a server implements a service, it registers to the service name with a certain name and possibly a brief description of the service. When a client wants a service it calls that service using its name at the service name that gives him the reference to the server that offers that service.

In the case of SOAP-based web services which also respect the SOA paradigm, three XML-based standards are needed to achieve the registration, the description and the transport operations offered thus:

- **UDDI (Universal Description, Discovery, and Integration)**, an independent platform register based on XML used as a tool for recording and locating web services based on SOAP model, designed to be interrogated by SOAP messages to provide access to WSDL documents.
- **WSDL (Web Service Description Language)**, a language based on XML; it is used as a tool for formal description of the operations provided by a SOAP web service.
- **SOAP (Simple Object Access Protocol)**, a protocol for exchanging structured information based on XML for the messages format and on protocols such as RPC and HTTP for messaging negotiations and transfer; it is used as communication protocol for making web service (SOAP model) requests and to obtain answers.

The operation of a SOAP application consists of three time points grouped in Fig. 2 in three steps:

Step 1. A particular service provider implements a method that wants to make it available on the web. The service provider registers the functionality of method M at a name register (Action 1 from Fig. 2). This is the first action of the SOA architecture that runs only once or if any functionality of method changes, is executed again.

Step 2. Another point of time is when a consumer wishes to access a particular method. The client will ask the name register (Step 2a) about a service that provides the wished method (Action 2 from Fig. 2). That register tells the client the service that offers the sought operation (Action 3 from Fig. 2). The client addresses to the service notified by the register (Step 2b) with the desire to receive details on how it may call the method M and how are the results provided (Action 4 from Fig. 2). The service returns to the client a document written in WSDL with the information required (Action 5 from Fig. 2). Step 2 (2a and 2b) runs only at the first call of the method or when the location or context was changed.

Step 3. The client asks the service every time it wants to execute the method M using for this a SOAP document (Action 6 from Fig. 2), and this one responds every time also through a SOAP document (Action 7 from Fig. 2).

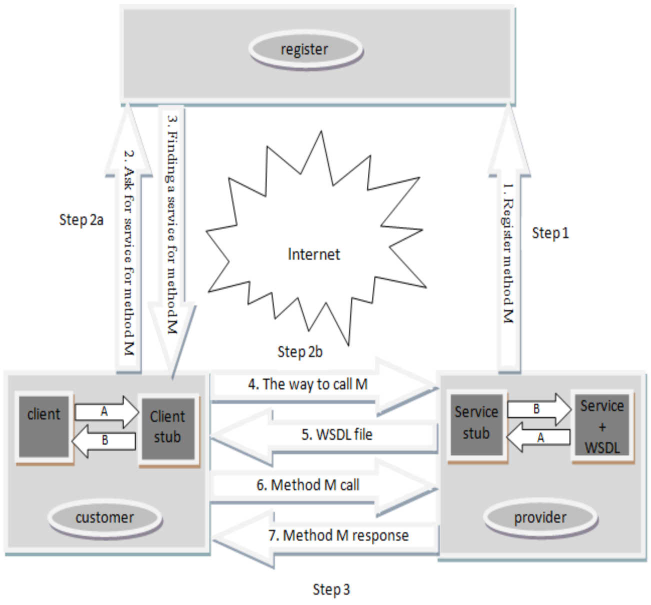


Fig. 2 The operation mode of SOAP web services and their clients

Just as in CORBA or RMI on the consumer machine and also on the service provider machine there is one stub component with the role to convert the data passing through them making serialization operations (getting data from local application and convert them into SOAP format - actions of type A in Fig . 2) and deserialization operations (getting data in SOAP format and convert them in local representation – actions of type B in Fig. 2).

Thus a communication between client and service consists of six steps, namely:

- The client sends to the client stub the necessary data to invoke the service by a local procedure call
- The client stub is required to convert the data in SOAP format and sends them to the service stub through the network
- The service stub converts the data from SOAP format in local format and sends them through a local procedure call to the service implementation
- The service tries to resolve the request and sends the response to the service stub also through a local call
- The service stub serializes the answer to the request and sends the SOAP document to the client stub
- The client stub deserializes the response data and sends the response by a local call to the client as the response to service request issued

Although a SOAP application may seem very complicated, everything related to SOAP and WSDL standards are hidden to the programmer under the stub of the client and of the service which in turn are generated automatically by specialized processors. All the platforms that implement such applications generate from the WSDL the stubs on the two machines: consumer and provider. The service stub is generated when the service is implemented (Action 1 in Fig. 2), and the client stub is generated before the first operation invocation (Action 2 in Fig. 2).

IV. REST Model

REST (Representational State Transfer) is a style of software architecture alternative to mechanisms like RPC (Remote Procedure Call) and SOAP-based web services. What Fielding summarized in his dissertation [9] was very successful, and REST became leader of the market for web services. REST is easier than SOAP and it also has a short content so it consumes less bandwidth. A REST web service has all the advantages of web service based on SOAP: is platform independent, language independent, can be used with firewalls and is based on standards (running over HTTP).

REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations. It uses POST for Create, GET for Read, PUT for Update and DELETE for Delete. REST requests can use POST instead of GET even for Read so there is no limit in length. REST borrowed CRUD requirements from databases. The most important example of a implementation of a system conforming to REST is the World Wide Web.

A RESTful web-service can be seen as a web-service implemented using HTTP and respecting the principles of REST. So, REST can be seen as a set of principles (rules for telling how standards like HTTP and URI can be used):

- Use URI for each resource which deserves to be specified
- Use links to refer the resources
- Use in a correct way the HTTP standard methods: GET, PUT, POST, DELETE so that the clients to be able to interact with resources by offered operations
- Use multiple representations of the resources for different needs

- Keep the state of the resource in the client side or make the resource a stateful resource if you want to keep the state

From this set of principles we can see that a REST-style based service is made of three parts: resources, representations of resources and self-descriptive messages. In fact REST Web-services are collections of resources, which have representations and can be accessed through HTTP methods.

This kind of services are collections of resources with the following three aspects:

- URIs which identify the resources (which also have the name REST objects or subjects)
- Representations of the resources which can have one or more representations commonly
- HTML, JSON, XML or YAML but they can be any other kind of valid media
- The set of operations supported by the web services using HTTP methods (POST, GET, PUT, DELETE), the operations performed by these methods are also called actions or predicates that are carried on objects (resource issues)

An action may be considered "safe" if it does not alter state performance context, so a data reading can be considered such an action. The only method considered safe by the REST is GET. An action may be considered idempotent if repeated its performance does not change the resource status. The methods considered idempotent by REST are: GET, PUT and DELETE.

The focus of the REST principle is the resource that is referenced by a global identifier, an URI in HTTP and is accessed by the network components (clients, servers, caches, etc.) through a standardized interface such as HTTP which allows the exchange of the resource representation between network components.

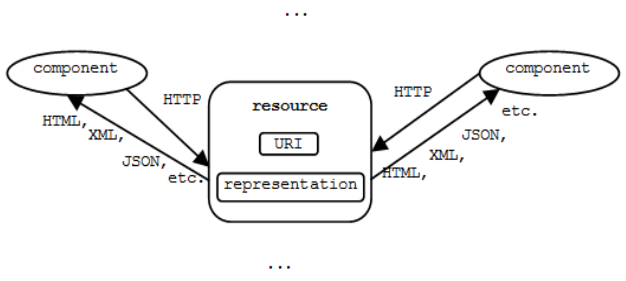


Fig. 3 REST Principle

According to the REST principle as shown in Fig. 3 any application (network component) can interact with a resource by knowing three things: resource identifier, the desired action (given in the URI) and the HTTP method that obtains the operation without needing to know anything about the link between resource and server resource store. To also understand the response from the resource, the application must also know a fourth thing: the format of the response from the resource which usually is HTML, XML, JSON, but can be any valid media type as an image, plain text or other content.

V. WSWrapper Model

WSWrapper is a wrapper who wants to follow the patterns of all three types of web services, namely XML-RPC, SOAP and REST each with its peculiarities. For this reason it was necessary to extract what is common to the three models, but also to respect the essence of each type of service separately.

In the previous chapters were presented by some suggestive figure the operating principles for the three types of web services. We will extract from these the model and the operating principle of WSWrapper.

WSWrapper can be seen as a function, namely:

$$WS : C \to S$$

where C is the set of customers which can use the existing web services, and S is the set of existing web services, principle met in the case of algorithms, namely:

$$A : D \to R$$

where D is the set of input data, and R is the set of output data (of results).

Going on the principle of algorithms we can see that customers from the WS domain actually have the input data that are converted into output data (results) by the services from codomain.

The function WS must respect all the principles of an algorithm, namely:

- Generality, meaning that the service can be accessed for any input data, for correct data to obtain desired results, and for wrong data to react with error messages.
- Finiteness, meaning that regardless of the input data for this web service is called to return the correct output data or error messages in a limited time.
- Uniqueness, meaning that for the same set of input data sends to the service several times the results received will be the same each time.

Following the principle of perceiving a web service as an algorithm and extracting the common of the three types of web services XML-RPC, SOAP and REST, a web service can be modeled as in Fig. 4.

The operation of a service implies the existence for a web service provider and at least one web service client that uses what the service offers.

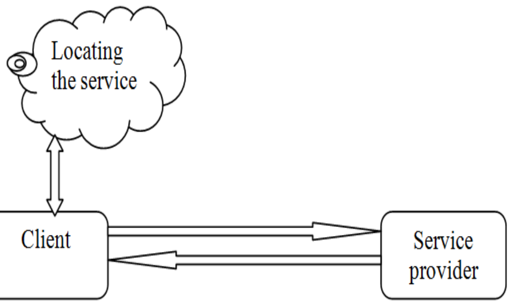


Fig. 4 The operating principle of web services

The service provider is:

- For XML-RPC model – the server where is the function offered by the service and the XML-RPC listener;

- For SOAP model - the server where is the function offered by the service, the service WSDL and the service stub;
- For REST model – the resource that provides the service identified by a URI and a representation of it.

The client of a service must know the location where it can access the service to be able to contact the provider.

The interface used for WSWrapper is quite simple as can be seen in Fig. 5 for the server and in Fig. 6 for the client. The wrapper has two interfaces (server-side and client-side) that work together to achieve abstraction in all of the 12 cases resulting from the combination language - service.

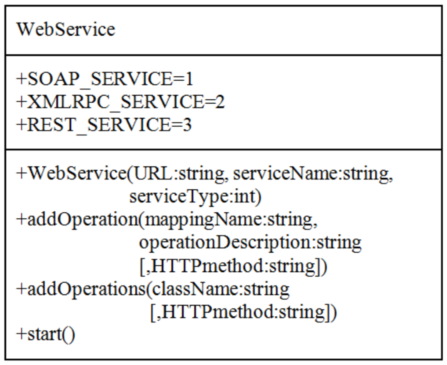
Server Web service is implemented in class WebService class that allows the programmer to define a web service.

The service can be obtained by calling the constructor of the class WebService which will be called with the name of the service (serviceName) and the URL (URL) where people can access the service and the kind of service (serviceType) which is one of the three models given by the three constant:

SOAP_SERVICE=1
XMLRPC_SERVICE=2
REST_SERVICE=3

The class contains two more methods for exposing web methods. addOperation will take two or three parameters as appropriate depending on the type of service. The first two parameters are needed for all three types of services referring to the mapping of a class method or mapping of a global function (where language permits it) and the complete path to the method implementation. The third parameter is optional and is used only if the service is of REST type and is the HTTP method with which the service can be call (GET, POST, PUT, DELETE, etc.). addOperations is a shortcut for avoiding more addOperation, to map all public methods from the class mentioned in the className parameter with HTTPmethod for the REST service.

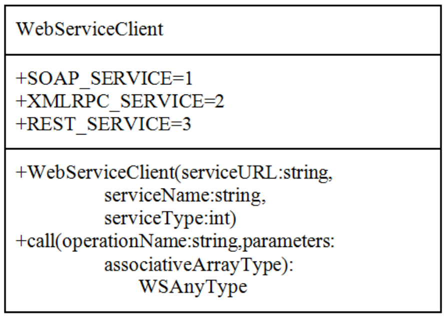
The server will be started with the method start() of WebService that will react differently depending on the type of service.



```
WebService
---------------------------------------------
+SOAP_SERVICE=1
+XMLRPC_SERVICE=2
+REST_SERVICE=3
---------------------------------------------
+WebService(URL:string, serviceName:string,
            serviceType:int)
+addOperation(mappingName:string,
              operationDescription:string
              [,HTTPmethod:string])
+addOperations(className:string
               [,HTTPmethod:string])
+start()
```

Fig. 5 The Service interface of the WS Wrapper

The web service client will just have a WebServiceClient class. A WebServiceClient object is obtained in the same way as the web service wrapper calling the constructor of the class which knows the host of the web-service server (in serverURL) and the name of the service (serviceName). On the client side, the serviceType parameter can take the same values as for the WebService server.



```
WebServiceClient
---------------------------------------------
+SOAP_SERVICE=1
+XMLRPC_SERVICE=2
+REST_SERVICE=3
---------------------------------------------
+WebServiceClient(serviceURL:string,
            serviceName:string,
            serviceType:int)
+call(operationName:string,parameters:
            associativeArrayType):
               WSAnyType
```

Fig. 6 The Client interface of the WS Wrapper

The interface has also a call method, used to call an exposed web-method which returns a WSAnyType object, a unification of types object and have two arguments, first for the name of the method exposed (operationName) and second for the parameters of the method (parameters). The parameters are passing into an associative array: Map Java, Hashtable C#, array PHP, Dictionary Python. All inherit a generic WSAnyType.

The WSAnyType belongs to the other interface of classes we have been talking about in [3].

VI. Uniform Interface in WSWrapper

WSWrapper framework exposes two public application interfaces: the first two interfaces define the unified WebService and WebServiceClient classes. There is another interface defining the types hierarchy (WSAnyType) used by the WebService and WebServiceClient.

A. Java web service interface for WSWrapper

```
 package ro.ubbcluj.cs.wsw;
 public class WebService {
  public static int SOAP_SERVICE  = 1;
  public static int XMLRPC_SERVICE = 2;
  public static int REST_SERVICE  = 3;
  public static WebService getWebService(String uRL,
              String serviceName, int serviceType) {
   if (serviceType == XMLRPC_SERVICE)
       return new WebServiceXj(uRL, serviceName);
   if (serviceType == SOAP_SERVICE)
       return new WebServiceSj(uRL, serviceName);
   if (serviceType == REST_SERVICE)
       return new WebServiceRj(uRL, serviceName);
  }
  public int addOperation(String mappingName,
           String operationDescriptor)
              throws Exception {return 0;}
  public int addOperation(String mappingName,
           String operationDescriptor, String hTTPmethod)
              {return 0;}
  public int addOperations(String operationDescriptor,
           String pClas)
               throws Exception{return 0;}
  public int addOperations(String className,
           String operationDescriptor, String hTTPmethod)
               {return 0;}
  public void start() throws Exception {;}
}
```

Source 1 The WebService class

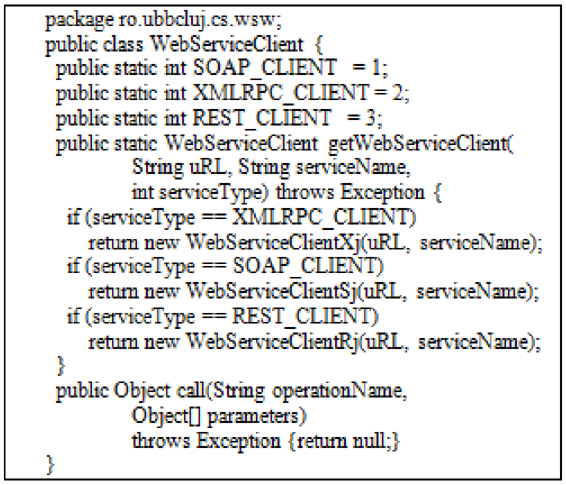
For exposing universal WebService operations, as defined in [3], we use for the main part of the Java services like in the Source 1.

For each web services: XML-RPC, SOAP and REST, an adequate open source implementation will be chosen. Using these distributions, three subclasses of WebService will be implemented, respectively: WebServiceXj, WebServiceSj and WebServiceRj. Each created class will implement based on the particular distribution, a constructor. Also, the methods addOperation, addOperations and start will be rewritten.

Our choice among all open source distributions was: Apache XML-RPC [21], Apache Cxf library [20], and RESTeasy [22] from JBoss distribution.

B. *Java web service client interface for WSWrapper*

For exposing WebServiceClient objects, as defined in [3], we use the implementation of the Java client like in the Source 2:



```
package ro.ubbcluj.cs.wsw;
public class WebServiceClient  {
  public static int SOAP_CLIENT   = 1;
  public static int XMLRPC_CLIENT = 2;
  public static int REST_CLIENT   = 3;
  public static WebServiceClient  getWebServiceClient(
             String uRL, String serviceName,
             int serviceType) throws Exception {
   if (serviceType == XMLRPC_CLIENT)
      return new WebServiceClientXj(uRL, serviceName);
   if (serviceType == SOAP_CLIENT)
      return new WebServiceClientSj(uRL, serviceName);
   if (serviceType == REST_CLIENT)
      return new WebServiceClientRj(uRL, serviceName);
  }
  public Object call(String operationName,
             Object[] parameters)
             throws Exception {return null;}
}
```

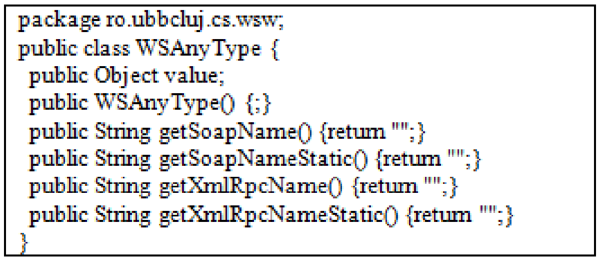
Source 2 WebServiceClient class

For each of the three types of web service clients: XMLRPC, SOAP and REST, an adequate open source distribution for implementation is used. It is not necessary to choose the same distributions as to the service part. For XMLRPC we choose the same distribution as on the service part: Apache XML-RPC [21]. For REST we use even our own distribution presented in [16] or RESTeasy [22] from JBoss distribution. For SOAP, we used two distributions: Apache Cxf library[20] and Apache AXIS 1.1 [23]. Using these distributions, analogously – but not mandatory the same distributions from the service part, three subclasses of WebServiceClient will be implemented, respectively: WebServiceClientXj, WebServiceClientSj and WebServiceClientRj. Each of these classes will implement, depending on the particular distribution, a constructor and will rewrite the method call.

C. *Data type for Java interface in WSWrapper*

To use the data types, the subclasses of the following code (Source 3) will be used.

The getSoapName() function returns the actual name for SOAP case, for example "xsd:int" for an integer value. Analogously, getXmlRpc() returns "int" or "i4" in the XML-RPC case. No method is necessary in the case REST.



```
package ro.ubbcluj.cs.wsw;
public class WSAnyType {
 public Object value;
 public WSAnyType() {;}
 public String getSoapName() {return "";}
 public String getSoapNameStatic() {return "";}
 public String getXmlRpcName() {return "";}
 public String getXmlRpcNameStatic() {return "";}
}
```

Source 3 WSAnyType class

After the service(s) and client(s) are implemented, a jar archive named WSWrapper.jar will be created. It will contain all the jars archives from the above defined distributions and the following classes: WebService, WebServiceXj, WebServiceSj, WebServiceRj, WebServiceClient, WebServiceClientXj, WebServiceClientSj, WebServiceClientRj.

VII. Implementations Solutions

We will present an implementation of a XML-RPC web service in Java and a REST web service client in Java.

A. *XML-RPC web service implementation example*

➢ *WebService class*

As mentioned above, the interface for a web-service is WebService which is a general class for all three types of service, where are calls for each class in part responsible for desired service (Source 1). In the case of XML-RPC implemented in Java this will be WebServiceXj which extends the WebService class.

The programmer will create a web service in the same manner regardless of the model used creating a WebService object as the return of the getWebService method, which is further responsible for using one of the three models REST, SOAP or XML-RPC.

In the method getWebService it is created a WebServiceXj object if we have a XML-RPC web service which needs two parameters: the url and the serviceName.

➢ *WebServiceXj class*

The class WebServiceXJ extends the class WebService (presented in Source 1), being the responsible class for XML-RPC web services in WSWrapper.

In the constructor of WebServiceXj class we obtain from the parameter which contains the URL namely uRL the port number on which the webServer object of type WebServer will listen. From this object is achieved with getXmlRpcServer method the service. It is set the configuration and the mapping for the service and finally the serviceName.

The WebServiceXj class is the class that connects the interface WebService unique for all three types of services and Apache XML-RPC package. Apache XML-RPC package is an open source package for XML-RPC web services development in Java.

```
package ro.ubbcluj.cs.wsw;
...
public class WebServiceXj  extends WebService {
 ...
 public WebServiceXj(String  uRL, String serviceName) {
  log.debug(serviceName);
  int dp = uRL.lastIndexOf(":");
  port = Integer.parseInt(uRL.substring(dp + 1,
                          uRL.lastIndexOf("/")));
  webServer = new WebServer(port);
  serviciu = webServer.getXmlRpcServer();
  config = new XmlRpcServerConfigImpl();
  serviciu.setConfig(config);
  mapping = new PropertyHandlerMapping();
  serviciu.setHandlerMapping(mapping);
  log.debug(""+mapping);
  this.serviceName = serviceName;
 }
```

Source 4 Constructor of WebServiceXJ class

The service is started with the start method. In the method we call the start method for the webServer object as shown in Source 5.

```
public void start() throws Exception {
  log.debug("");
  webServer.start();
 }
```

Source 5 Method start of class WebServiceXJ

➢ *The webservice class*

An particular application service, for an XMLRPC service, must contain the code like in the Source 6:

```
package p;
import ro.ubbcluj.cs.wsw.*;
 ...
public class ApplicationClassService {
 ...
  WebService service =
     WebService.getWebService(urlHostPort,    pathServ,
WebService.XMLRPC_SERVICE);
  service.addOperations(pathServ,
"p.ApplicationClassService");
  ...
  service.start();
 }
 ...
}
```

Source 6: A snippet from the Application class service

For creating and using the service, we need to compile the classes. In Windows, we use the following command:

javac –cp .;WSWrapper.jar ApplicationClassService.java

B. REST web service client implementation example

The solution of implementing a webservice client in Java using the wrapper WS-Wrapper will be a quick and easy one because all that the programmer needs to know is the client WS-Wrapper interface that is to be the same regardless of language or web service model used. This involves creating a single class and in its constructor it creates an WebServiceClient object on which will call the method call using the current parameters depending on the web service that wants to use.

In the following sections we want to go inside of WS-Wrapper and see how it hides the details of RestClient distribution presented in [16] to the programmer.

➢ *WebServiceClient class*

As mentioned above, the interface for a web-service client is WebServiceClient which is a general class for all three types of service, where are calls for each class in part responsible for desired service (Source 2). In the case of REST this will be WebServiceClientRj which extends the WebServiceClient class.

At this point it can be seen the power, the importance and the necessity of this wrapper which consists mainly of the fact that the programmer will create a web service client in the same manner regardless of the model used creating a WebServiceClient object calling the getWebServiceClient method, which is further responsible for using one of the three models REST, SOAP or XML-RPC.

In the method getWebServiceClient it is created a WebServiceClientRj object which needs two parameters: the url and the serviceName.

➢ *WebServiceClientRj class*

In the constructor of the WebServiceClientRj class the serviceName and the url are set and then it is started the service implemented in RestClient package presented in [16] as shown in Source 7.

```
public WebServiceClientRj(String uRL,
                String serviceName) throws Exception {
    this.serviceName = serviceName;
    this.url=uRL;
    serv=new RestClient(new URL(uRL+serviceName));
 }
```

Source 7 Starting the service

The constructor of the WebServiceClientRj class is the function that connects to and also hides details of a distribution for a type of webservice. Thus the webservice client programmer does not need to know the specifics of the distribution he uses. All that the programmer sees is a unified interface for all webservices and for all four languages used in the wrapper. This constructor is the one who actually uses the chosen distribution to be used to create a webservice client or a webservice server. For the Java language we used a new distribution created with the thought of this wrapper because the distributions on the market are too large.

The RestClient class requires a constructor with one parameter of URL type that is composed from the webservice url concatenated with the service name which contains within it both the class name and the function name that implements the service.

The call of the available methods at the web-service server is made of the call function from the WebServiceClientRj class by calling the call method from RestClient as shown in Source 8.

```
public Object call(String operationName,
                    Object[] parameters) throws Exception {
  serv.setMethod(operationName);
  HashMap<String,String> params=
                           new HashMap<String,String>();
  serv.call(operationName,
    new URL(url+serviceName),params,"","","text/html");
  return new Object();
}
```

Source 8 Calling a method

Because in REST model of webservice the name of the called operation is included in the url passed in the call and because in REST model (different from the other two models of webservices) is required the type of the method (GET, POST, DELETE, etc.) with which the operation is called, the operationName argument of the call function will be used to transmit that type and not the actual name of the function called as in the other two models of webservices.

➢ *The webservice client class*

The programmer who wants to implement a webservice client using the wrapper WS-Wrapper has now a pretty easy task, namely to implement a class where to create the object serv of type WebServiceClient by calling the method getWebServiceClient of the class WebServiceClient. Next for this serv object created before it calls the call method with two parameters: the first one which mentions the method type namely GET, POST, etc. and the second one which mentions the method parameters as shown in Source 9.

```
public RestClie(String urlHostPort, String pathServ)
                              throws Exception {
   WebServiceClient serv =
     WebServiceClient.getWebServiceClient(urlHostPort,
      pathServ, WebServiceClient.REST_CLIENT);
   Object[] param= new Object[0];
   Object resp = serv.call("GET",  param);
}
```

Source 9 RestClie contructor

The method type (GET, POST, etc.) is required only in the REST model, in other models it is not necessary so that the first parameter is used to transmit the actual name of the method which implements the web service.

In the REST model the effective method name is passed in the parameter serviceName which is made for this model from a concatenation of the filename where the method is located and the method name. This little trick is necessary to maintain uniformity for all three models of webservices for all four languages. In REST model the WSAnyType object returned by the call method will not be used but also appears for the uniformity.

VIII. Conclusion

The WebService interface and WebServiceClient interface are the same for all three types of services so that the programmer of a server or client web-service will only need to know the specifications for these interfaces regardless of the type of services: SOAP, XML-RPC or REST, interfaces that hide the specifics of each type.

For a REST web service in Java, the interface WebService will call a REST specific class namely WebServiceRj and the interface WebServiceClient will call WebServiceClientRj.

For a XML-RPC web service in Java, the interface WebService will call a XML-RPC specific class namely WebServiceXj and the interface WebServiceClient will call WebServiceClientXj.

Also for a SOAP web service in Java we have the class WebServiceSj which is called from the interface WebService and class WebServiceClientSj which is called from the interface WebServiceClient.

These six classes that are instantiated in WebService respectively WebServiceClient interfaces are responsible for hiding the details of each open source distributions used in WSWrapper.

We also want to offer in WS-Wrapper the posibility to choose from several free distributions to be used in implementing the web service or web service client.

We also want to add a new facility to WSWrapper namely WSGenerator. WSGenerator was introduced because of the lack of a uniform, platform independent and automated proxy generation library. The intention of WSGenerator is to simplify the development phase of a distributed application.

WSGenerator provides a simple and easy to use solution for Web Service proxy generation. The main advantage of this component is the uniform and platform independent interface.

WSGenerator provides to the end user a simple and easy to use tool for generating client Web Service proxies. The proxy generation library offers a unified set of solution for all the three major Web Service types: RPC, SOAP and REST. By using this component, generating a proxy for a given service become easy. The Web Service specific details and operations, including specific transformations remain hidden to the end-user.

References

[1] S. Allamaraju, RESTful Web Services Cookbook, O'Reilly, 2010
[2] J. Bean, SOA and Web Services Interface Design, Principles, Techniques and Standards, Elsevier, 2010
[3] F. Boian, D. Chinces, D. Ciupeiu, D. Homorodean, B. Jancso, A. Ploscar, WSWrapper – A Universal Web Service Generator, Studia Univ. Babes-Bolyai, Informatica, An LV, 2010, no. 4, pp 59-69.
[4] F.M. Boian, Servicii web; modele, platforme aplicatii, Ed. Albastra, Cluj, 2011.
[5] F.M.Boian, Unification of the Web Services, Proceedings "Zilele Academice Clujene 2010 (ZAC2010)", Ed. Presa Universitară Clujeană, Cluj 2010, ISSN 2066-5768, pp.92-97.
[6] N. David, C.G. Carstea,.I.Gh. Ratiu, L. Patrascu, D. Damian, Web Services – Opportunities and Challenges, Proceedings of the 10th WSEAS International Conference on Applied Computer Science (ACS '10), Japan, 2010, ISSN: 1792-4863, ISBN: 978-960-474-231-8,pp.436-439
[7] H. M. El-Bakry, A. M. Riad, Q. F. Hassan, A. E. Hassan, N. Mastorakis, Technology and Recent Development of XML Web Services, Proceedings of the 4th WSEAS International Conference on

Business Administration (ICBA '10), Greece, 2010, ISSN: 1790-5109, ISBN: 978-960-474-161-8,pp.110-136
[8] H. M. El-Bakry, N. Mastorakis, Studying the Efficiency of XML Web Services for Real-Time Applications, Proceedings of the 2nd WSEAS International Conference on Sensors and Signals (SENSIG '09) Proceedings of the 2nd WSEAS International Conference on Visualization, Imaging and Simulation (VIS '09) Proceedings of the 2nd WSEAS International Conference on Materials Science (MATERIALS '09), USA, 2009, ISSN: 1790-5117, ISBN: 978-960-474-135-9, pp.209-219
[9] R.Fielding, Architectural Styles and the Design of Network-based Software Architectures - Dissertation, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
[10] D. Homorodean, F.M. Boian, SOAP Web-Services in Python: Problems and Solutions. Proceedings "Zilele Academice Clujene 2010 (ZAC2010)", Ed. Presa Universitară Clujeană, Cluj 2010, ISSN 2066-5768, pp. 105-111.
[11] F. Ismaili, B. Sisediev, Web Services – Current Solutions and Open Problems, Proceedings of the 8th WSEAS International Conference on Applied Informatics And Communications (AIC'08), Greece, 2008, ISSN: 1790-5109, ISBN: 978-960-6766-94-7, pp.115-119
[12] B. Jancso, RESTful Web Services. Proceedings "Zilele Academice Clujene 2010 (ZAC2010)", Ed. Presa Universitară Clujeană, Cluj 2010, ISSN 2066-5768, pp. 158-163.
[13] M. Kalin, Java Web Services; Up and Running, O'Reilly, 2009
[14] S. St. Laurent, J. Johnson, E. Dumbill, Programming Web Services with XML-RPC, O’Reilly, 2001.
[15] E. Newcomer, Understanding Web Services; XML, WSDL, SOAP, and UDDI, Addison Wesley, 2004
[16] A. Ploscar, A Java implementation for REST-style client web service, Proceedings "Zilele Academice Clujene 2010 (ZAC2010)", Ed. Presa Universitară Clujeană, Cluj 2010, ISSN 2066-5768, pp. 140-146
[17] L. Richardson, S. Ruby, RESTful Web-Services, O’Reilly, 2007.
[18] S. Tanasa, C. Olaru, S. Andrei, Java de la 0 la expert, Polirom, Iaşi, 2007
[19] D. Tidwell, J. SNall, P. Kulchenko, Programming Web-Services with SOAP, O’Reily, 2001.
[20] * * * apache-cxf-2.2.7. http://axis.apache.org
[21] * * * apache-xmlrpc-3.1.3. http://ws.apache.org/xmlrpc
[22] * * * RESTEasy JAX-RS: RESTFul Web Services for Java. http://jboss.org/resteasy
[23] *** http://axis.apache.org
[24] *** The Java EE Tutorial Sun Microsystems, 2008, http://download.oracle.com/javaee/6/api
[25] ***, JavaTM Platform Enterprise Edition, v 5.0 API Specifications
[26] *** http://en.wikipedia.org/wiki/Representational_State_Transfer
[27] *** http://en.wikipedia.org/wiki/Web_service
[28] *** http://flexria.wordpress.com/2008/08/01/web-services-and-php-soap-vs-xml-rpc-vs-rest
[29] *** http://ws.apache.org/xmlrpc
[30] *** http://www.infoq.com/articles/rest-introduction A Brief Introduction to REST
[31] *** http://www.javapassion.com
[32] *** http://www.javaworld.com/jw-01-2000/jw-01-howto.html