Rest Web Services in Java Using WS-Wrapper

PLOSCAR ADINA
Faculty of Natural Sciences, Engineering and Computer Science
“Vasile Goldis” Western University
Arad 310025, Bd. Revolutiei nr. 94-96
ROMANIA
adina_ploscar@yahoo.com

Abstract: - In the software market, the distributed software is increasing and unfortunately developers create different packages, each with its own specifics. Therefore, together with a team of researchers, we have tried to unify the various web services technologies implemented in the four most important languages of the moment in a unified library, a wrapper that we called WS-Wrapper. My part of the project which is also the subject of this paper is implementing a REST web service and also a REST client under Java platform. WS-Wrapper is trying to meet the needs of each Web service models because each of the three models of web services (SOAP, XML-RPC, REST) have their own characteristics. REST needs to know by which service method (GET, POST, PUT, DELETE, etc.) is called, while others models do not need this. This paper will address the needs of a general REST application and especially on the Java platform using WS-Wrapper.

Key-Words: - web-services, wrapper, REST, client, Java, method

1 Introduction

In this moment we can distinguish three models of Web services: XML-RPC, SOAP and REST that benefit of many libraries either free or fee for most modern programming languages available on the market. Since each of the developers of the libraries for creating these types of services enter its specific in creation of the library packet, a programmer who wants to use one of the three models of services in a particular programming language has to learn the specific of the packet gave by each of the developers so he needs a lot of time for every different library.

WS-Wrapper comes as a solution to reduce this time studying a single package (that wrap all the web-services types) which offer the possibility of creating web services and clients of these services in any of the currently three existing models in four modern programming languages, namely Java, C#, PHP and Pyton.

This wrapper will provide a simple interface for creating web service clients and servers and will also contain all the libraries needed for the web services, so the programmer do not have to take care of downloading and installing the necessary packages of services in various models of various programming languages.

The interface used is quite simple as can be seen in Fig.1 for the server and in Fig.2 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.

<table>
<thead>
<tr>
<th>WebService</th>
</tr>
</thead>
<tbody>
<tr>
<td>+SOAP_SERVICE=1</td>
</tr>
<tr>
<td>+XMLRPC_SERVICE=2</td>
</tr>
<tr>
<td>+REST_SERVICE=3</td>
</tr>
<tr>
<td>+WebService(URL:string, serviceName:string, serviceType:int)</td>
</tr>
<tr>
<td>+addOperation(mappingName:string, operationDescription:string [,HTTPmethod:string])</td>
</tr>
<tr>
<td>+addOperations(className:string [,HTTPmethod:string])</td>
</tr>
<tr>
<td>+start()</td>
</tr>
</tbody>
</table>

Fig.1 The Service interface of the WS Wrapper

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 gave 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.

<table>
<thead>
<tr>
<th>WebServiceClient</th>
</tr>
</thead>
<tbody>
<tr>
<td>+SOAP_SERVICE=1</td>
</tr>
<tr>
<td>+XMLRPC_SERVICE=2</td>
</tr>
<tr>
<td>+REST_SERVICE=3</td>
</tr>
<tr>
<td>+WebServiceClient(serviceURL:string, serviceName:string, serviceType:int)</td>
</tr>
<tr>
<td>+call(operationName:string, parameters: associativeArrayType): WSAnyType</td>
</tr>
</tbody>
</table>

Fig.2 The Client interface of the WS Wrapper

The web service client will just have a WebServiceClient class. A WebServiceClient object is obtained in the same fashion 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.

The interface has also a call method, used to call an exposed web-method which returns a WSAnyType object, a unified 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). 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 [1], but is not relevant for the REST web services and that is why we will not go into detail about it here.

2 Problem Formulation

REST (Representational State Transfer) is a style of software architecture alternative to mechanisms like RPC (Remote Procedure Call) and SOAP-based web services.[3] REST is easier than SOAP and it also has a short content so it consumes less bandwidth. 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. 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 deserve 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 representation 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.

![Diagram of REST Principle](image)

**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 need 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.

There are different libraries for implementing REST web services in all of the four languages mentioned above that are used in WS-Wrapper. But in this paper we will refer to the REST model by the perspective of Java. After studying several distributions for REST Java web-services, we stopped at RESTeasy JBoss, but in parallel with using this distribution we began implementing our own distribution that is more simple [4]. At this point the distribution is made only for the client and that package will be used in WS-Wrapper for a REST Java web-service client. For server-side Java REST web services we will use temporary RESTeasy JBoss distribution.

In [1] can be seen the REST model in the other three languages and thus can be noticed the huge differences in programming languages and libraries for this type of web service and of course the necessity of WS-Wrapper.

### 3 Problem Solution

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 [4] to the programmer.

#### 3.1 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 (Fig.4). In the case of REST this will be WebServiceClientREST 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 `WebServiceClientREST` object which needs two parameters: the `url` and the `serviceName`.

### 3.2 WebServiceClientREST class

In the constructor of the `WebServiceClientREST` class the `serviceName` and the `url` are set and then it is started the service implemented in `RestClient` package presented in [4] as shown in Fig. 5.

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

Fig. 5. Starting the service

The constructor of the `WebServiceClientREST` class is the function that connects to and also hides details of a distribution for a type of webservice so that the webservice client programmer does not need to know the specifics of the distribution he use. 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 type `URL` 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 webservice server is made of the call function from the `WebServiceClientREST` class by calling the call method from `RestClient` as shown in Fig. 6.

```java
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();
}
```

Fig. 6. 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 `serviceName` 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.

### 3.3 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 Fig. 7.

```java
public RestClient(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);
}
```

Fig. 7 RestClic constructor

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 is the method and the method name.

References:

4 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, the interfaceWebService will call a REST specific class namelyWebServiceRest andWebServiceClient will call WebServiceClientREST.

In the future we want to make our own distribution for the server side of a REST web-service in Java as we did for the client side and then we want to use this distribution in WS-Wrapper project.

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