STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LV**, Number 4, 2010

WSWRAPPER — A UNIVERSAL WEB SERVICE GENERATOR

FLORIAN BOIAN, DIANA CHINCEŞ, DAN CIUPEIU, DAN HOMORODEAN, BEATA JANCSO, AND ADINA PLOSCAR

Abstract. The need for distributed software applications is increasing day by day. Having to choose from a large variety of libraries, and to learn what each is capable of and how to use them is time consuming and overall can decreases the productivity of an engineering team. We created the WS Wrapper as a unified library on top of existing language-specific libraries, to transparently solve all dependencies, and to provide the developer with a solution that can be used in a distributed application without having to know what happens behind the scenes.

1. Introduction

Looking to the software industry nowadays, you can see that distributed applications have become a requirement. Before even thinking how a project can be developed, a good designer needs to decide what technologies should be used. In recent years, the one technology that has gained popularity is web services [1, 7, 9, 10], but even though it provides a very good design to a project, in most situations (programming languages) you can find that these are very hard to use. We created a web-services wrapper on top of existing libraries that unifies these technologies, such that they can be easily used from several programming languages. This wrapper will be available in: C# [4, 14, 20], Java [19, 11, 18], PHP [16, 19, 9] and Python [13, 17, 15] for all types of web-services: XML-RPC, SOAP, REST.

In the following sections, we are presenting the state of art in web services, what a user must do to use all types of web-services in the previously mentioned

Received by the editors: November 5, 2010.

2010 *Mathematics Subject Classification.* 68N25, 68U35.

1998 *CR Categories and Descriptors.* C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks – *Distributed systems*; D.2.12 [**Software**]: Software Engineering – *Interoperability*; H.3.5 [**Information Systems**]: Information Storage and Retrieval – *On-line Information Services*; I.2.2 [**Computing Methodologies**]: Artificial Intelligence – *Automatic Programming*.

Key words and phrases. Web services, system specification, program generation, frameworks.

programming languages, and how this wrapper simplifies the process. We are also giving simple examples, just to illustrate how different it is to implement web-services in the four chosen programming languages.

The main advantage of the WS Wrapper is the simple and uniform access to the underlying web services.

2. Web Service Models

2.1. **XML-RPC.** A remote procedure call is call made by a program, over the network, to a procedure executing on a remote machine connected. Let's say that program B has implemented a *date* function and program A has not. Instead of just implementing it, assuming this would take a lot of time, program A decides to call program B's function. The function call is encoded with XML and sent over HTTP requests.

Of course, this is a very simple example. Let's say that program B has a more complex function, which needs arguments, such as the summing of two numbers.

The data types known by the XML-RPC [7] protocol are as follows: array, base64, boolean, date/time, double, integer, string, struct, nil. We would like to point these out, just to see the differences between the three types of web-services and the need for unification.

For demonstrating how XML-RPC works in the chosen programming languages, we have selected the following libraries to be used:

- C# - XmlRpcCS [20];
- Java – Apache XML-RPC [12];
- PHP – xmlrpc distribution by Dumbill [19];
- Python - xmlrpclib [17].

In C#, for using the mentioned library, we had to compile all the source files in the first place. After this, the library had to be loaded into memory, and from this moment the flow of creating a new web service is similar to the one in Java, meaning that all the public methods of a specific class are exposed as web-methods. Classes are exposed as web-services through XmlRpcServer.The client creates a XmlRpcRequest object, which is used for calling web service methods.

In Java, the client creates a XmlRpcClient object, an object of parameters that is passed to the method, and calls it. On the server side, a WebServer object is created, and to this object the user may add Java classes that will expose all their public methods as web methods. We give all the names details in this article, just to outline how many differences there are in using web-services.

In PHP, an xmlrpc_server object needs to be created, and pass to it the list of web-methods that will expose. Each web method needs to have a name, an implementation and a signature. Each web method needs to decode and encode its arguments. On the client side, we have to create an xmlrpc_client object and send xmlrpcmsgs through it.

Last, but not least, in Python a SimpleXMLRPCServer needs to be constructed, which will register the web-methods that will be exposed. To call the web-methods, the Python client needs to construct an xmlrpclib.ServerProxy and call the methods using the getattr function.

It is evident from the above, that using XML-RPC in these four programming languages can be quite different, and we have only pointed out the high level differences.

2.2. **SOAP.** SOAP[10] is another protocol which can be used in exchanging information between two applications, by calling web-service methods. SOAP sends Envelopes, which are XML messages having a well-defined structure. The "low" level protocol used for sending these messages is HTTP, same as it is for XML-RPC. One of the main features in using this protocol in implementing web-services is that, using SOAP, one can publish the web service description, using WSDL (Web-Service Description Language). [1,10]

So basically, the flow in this situation is:

- Program A wants to call a web-method from program B;
- Program A looks at the WSDL of program B;
- Program A chooses the method;
- Program A calls the method using SOAP envelops;
- Program A receives the answer in an envelope.

It is very simple to develop SOAP servers in C#[4]. You just need to create an asmx file and place this in IIS. The strange thing here is that three very similar things need to be done so that the service is deployed:

- The first line in the file must be a WebService tag;
- The web-service class needs to be annotated with WebService;
- The web-service class needs to extends WebService.

Implementing SOAP clients in C# just requires the wsdl executable, which is shipped with the .NET framework. The URL of the WSDL needs to be given to this executable as a command line argument, and the C# stubs are generated. After this, all the files need to be compiled and grouped in a library which will be used by the client at runtime.

In Java[11], for implementing SOAP web-services we need the Apache Cxf library. Similarly, but simpler than C#, Java classes will just need to

be annotated with WebService.Each method that will be exposed as a web-method will have the WebMethod annotation. After doing so, an Endpoint needs to be defined, action which will actually publish the web-service. Using the apt executable, the needed .class files for the web-service will be created.

Client-side, wsimport generates the needed stubs by specifying the web-service's WSDL. As the C# wsdl executable, wsimport generates source files, too, which afterwards need to be compiled and added in the CLASSPATH.

Using SOAP in PHP requires NuSOAP [16],which is one of the first libraries that were developed in this programming language. The basic flow is that a nusoap_server is created, to which the web-methods are registered. The client constructs a nusoap_client,and uses it to call the methods that were registered with the server.

Python's list of dependencies and requirements for running a SOAP server and client are a bit longer than the other's programming languages [13]. We need to mention that the end-user would have to manually download the exact version of libraries. This being said, the list of dependencies is: Python 2.6, PyXML 0.8.4, lxml 2.2.2, soaplib, pytz, and suds 0.3.9.

For the SOAP server, each web method is decorated with @soapmethod. The arguments of this decorator have to be a list of types that are accepted as method arguments plus the return type of the method. The Python client uses the suds library and just creates a suds.Client object, used afterwards for web-method calls.

As in the case of XML-RPC, we can see the same pattern for SOAP. Even though we could group these techniques in two by similarity: (Java, C#) and (Python, PHP), there is a great difference even between the elements in the same group.

2.3. **REST.** REST (Representational State Transfer)[9] is a style of software architecture. The best known implementation of a system conforming to REST is the World Wide Web. A RESTful web-service is a web-service implemented using HTTP and the principles of REST. Web-services that are implemented using REST are seen as a collection of resources, which can be accessed through HTTP methods.

Implementing REST web-services in C#[14] is quite simple. As for SOAP, all you need is the *asmx* file containing the web-service and web-methods, file that will be placed in IIS. The class needs to extend HTTPRequest and ProcessRequest method will decide which method will be called for each type of requests (GET, PUT, POST, DELETE). For the REST client, it just has to create a HTTPWebRequest object which will be used to call the web-methods.

Using the RESTeasy [18] JBoss distribution, for exposing REST Java web-services, firstly the class that will expose web-methods needs to be annotated with @Path, indicating the path with which the web-service can be accessed. Each web method has four annotations: @Path, indicating the path for the method, @Produces, indicating what type of data the method produces, @Consumes, indicating the input data type for the method, and the HTTP method. The Java REST client is a straightforward HTTPURLConnection, setting to it the content type and the HTTP request method.

The best PHP solution that can be used to implement REST web-services are the wrappers implemented by DaSilva[3], which are RestClient and RestServer objects. The RestServer allows the user to add methods that will be exposed as web-methods, and the RestClient allows the user to call any of the web-methods, using one of the available HTTP-methods: GET, POST, PUT, DELETE.

CherryPy[13] is the REST Web Service library for Python we chose to use out of several existing libraries. These REST Web Service libraries often are difficult to use, and in the majority of cases it is easier to implement a new Python module for REST Web Services. The Python server part needs to extend the http.server.BaseHTTPRequestHandler module, providing implementation for the four HTTP methods (GET, POST, PUT, DELETE). For the Python client, things are even simpler, meaning it just has to open an http.client.HTTPConnection and call the methods using the right HTTP-method.

We can notice the huge differences in programming languages and libraries for this type of web-service, too.

3. WS-Wrapper

Knowing the current situation, the difficulty to implement web-services in different programming languages, we thought of a solution that would simply wrap all the web-services types in these four programming languages.

As the necessity of such a solution night not be evident, we present in the following the main motivation for this work. In [5, 6, 8] we presented some preliminary considerations on particular platforms. The first reason that could come to our mind was to simplify the process one needs to follow when starting writing web-services. The WS-Wrapper not only would offer the user a nice and simple interface, which he can use to write web-services clients and servers, but would also make sure that the system he is using is compliant, meaning it has installed all the required libraries.

One of the main features of this project will be, as we have mentioned in the previous paragraph, checking that the system on which web-services are

ran is compliant. For each web-service type (XML-RPC, SOAP and REST) and each programming language, there will be scripts/installers that will make sure that all the packages required for the web-services are available, and if not it would download/install them.

The web services wrapper that we present exposes two application interfaces that work seamlessly together to provide the same high level of abstraction on each of the twelve possible language-service combinations. First interface contains the wrapper classes for the unified web service and web client. This interface is shown in Figure 1. The Client interface is presented in Figure 2. Some details of Figures 1 and 2 need to be exposed:

- serviceType and clientType are one of the integer constants defined in class.
- operationDescriptor is one of:
  - Class::methodname – for referencing a public method in a class
  - globalFunction – where it is permitted by the language
- addOperations mapping all public methods from class (a shortcut foe avoid more addOperation)
- HTTPmethod will use only for REST_service
- start starting the service
- parameters are passing into an associative array: Map Java, Hashtable C#, array PHP, Dictionary Python. All inherit a generic WSAnyType.

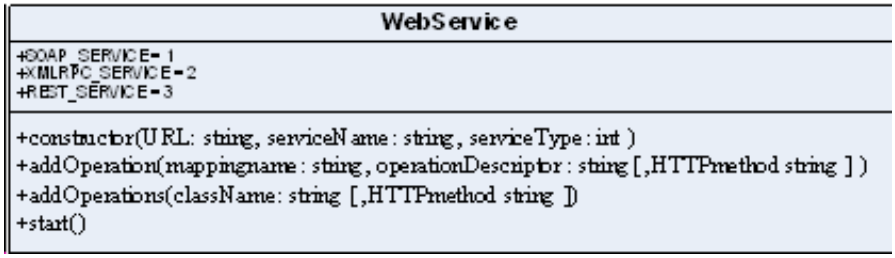

FIGURE 1. The Service interface of the WS Wrapper

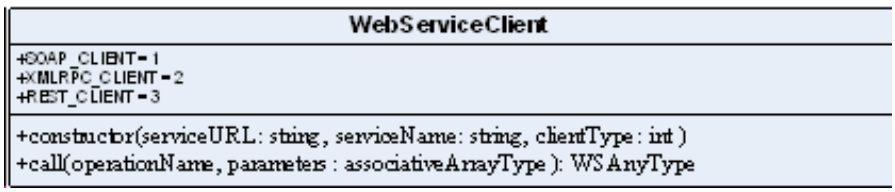

FIGURE 2. The Client interface of the WS Wrapper

The WSAnyType belongs to the other interface of classes we have been talking about and will be discussed later. So a web service object will be obtained by calling the constructor of the WebService class and specifying the URL of the service is mapping itself to, then giving a service name, and at last chosing one of the three constants (or public attributes) in the WebService class to obtain the desired type of web service. The class provides mapping methods for three types of operation mapping: mapping of a class method, mapping of a global function (where language permits it), and, for heavy services where service operations are defined in several classes, there is a method that adds to service exposed operations all public methods in a class.

The web service client is just a thin wrapper over consecrated web service clients presented in previous sections of this article. The client wrapper object is obtained in the same fashion as the web service wrapper, and the actual calling of an operation is done through the call() method, as explained in the diagram.

The second interface of objects that makes up WS Wrapper is the wrapper types' hierarchy. This hierarchy was designed to comply with all three web service types: REST, SOAP and XMLRPC. Also it is compatible with all four programming languages chosen and takes advantage of object oriented paradigm present in different forms in each of them. The diagram of this hierarchy can be observed in Figure 3. Two remarks:

- *getSoapName*() 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.
- wrap and unwrap methods are static and implemented in the base class for all structs. This means that all wrapping and unwrapping is already done for end user and adding a new struct type is as simple as declaring a class with need attributes, setters/getters and a constructor with suitable parameters. Every implementation of WSStruct in the four languages makes associations between struct class attributes and values in associative array based on the declaration of the actual class. For example class Person, declaring a struct with a person's data, is given.

4. An example

We have chosen to show the difference in implementing a simple web-service, Exec[2], in the old-fashioned way and using the WS Wrapper. This service exposes three methods:

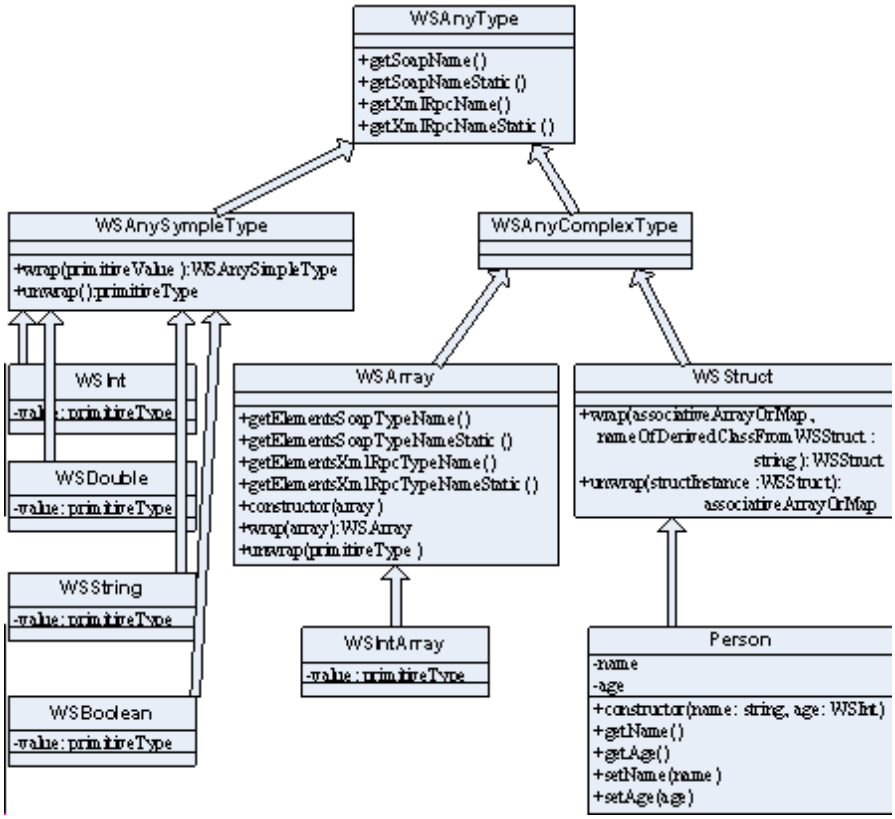

FIGURE 3. WSType classes' hierarchy

- ping()−−returns an identification string of the machine on which the service runs. This string contains the technology used for implementing the service, the name of the machine, the IP address of the machine and the current date and time of the machine.
- upcase(s)−−given as an argument the string s, this method return the same string, but transforming each letter in upper-case.
- add(a,b)−−given as argument two integer numbers a and b, the method returns the sum of a and b.

We present below brief specifications in WSWrapper of XML-RPC, SOAP, and REST services. For REST, some special specifications are necessary. The conventions we establish for implementing Exec follow the REST principles: having a fixed URL for the actions and creating mappings between HTTP methods and the exposed methods. After this service would be implemented, we can show five different methods of calling its exposed methods [2]:

URI	**HTTP method**	**HTTP body**	**Remarks**
http://address/ping	GET	-	No arguments for calling the method
http://address/upcase?s=black	GET	-	The string that has to be converted is passed as value of argument s.
http://address/upcase/black	POST		The string that has to be converted is passed as part of the URI.
http://address/add	PUT	a=66 & b=75	The two integers (66 and 75) are passed as the HTTP request body.
http://address/add/66/77	DELETE		The values of the integers to be added are part of the URL.

For example, we intended to use Java for services on the local host at the port 5678, and the class Implement has the static methods ping, upcase, and add. The services (for all three types) will be declared as follows:

```
WebService servX = new WebService("http://localhost:5678", "ExX",
                                  XMLRPC_SERVICE);
WebService servS = new WebService("http://localhost:5678", "ExS",
                                  SOAP_SERVICE);
WebService servR = new WebService("http://localhost:5678", "ExR",
                                  REST_SERVICE);
```

For mapping the implements of the methods, we specify:

```
servX.addOperation("ping", "Implement.ping");
servX.addOperation("upcase", "Implement.upcase");
servX.addOperation("add", "Implement.add");

servS.addOperation("ping", "Implement.ping");
servS.addOperation("upcase", "Implement.upcase");
servS.addOperation("add", "Implement.add");

servR.addOperation("ping", "Implement.ping", "GET");
servR.addOperation("upcase\?s=.+", "Implement.upcase", "GET");
servR.addOperation("upcase/.+", "Implement.upcase", "POST");
servR.addOperation("add", "Implement.add", "PUT");
servR.addOperation("add/-?[0-9]+/-?[0-9]+", "Implement.add",
```

```
                    "DELETE");
```

Remarks for REST: it is mandatory to specify at the name of the mapping, the prototype of the title of the URL. For this, it is necessary to use a regular expression [2,3].

After that, the service must be started, with servS.start(), servX.start(), servR.start().

Now, on the client part we intent to use PHP. First we must declare a proxy object to represent the service:

```
WebServiceClient $servX = new WebServiceClient
  ("http://TheServiceAddress:5678", "ExX", XMLRPC_CLIENT);
WebServiceClient $servS = new WebServiceClient
  ("http://TheServiceAddress:5678", "ExS", SOAP_CLIENT);
WebServiceClient $servR = new WebServiceClient
  ("http://TheServiceAddress:5678", "ExR", REST_CLIENT);
```

Finally, we can call the service methods as follows:

```
$servX->call("upcase", (array("s"=>"black");
$servS->call("ping", (array());
$servR->call("add", array("a"=>66,"b"=>75));
$servR->call("add/66/75", array();
```

5. Conclusions and Future work

Each of the technologies covered by the WS-Wrapper offer their own library for implementing web-services. We have presented above a solution that offers a unified approach to developing web services on all these platforms. To our knowledge, such unification has not yet been attempted in the existing art.

Currently, a team of PhD, MS, and BS students is working on implementing WS generators using the model from this paper. They will implement twelve projects, for three Web service technologies: XML-RPC, SOAP, REST and the programming languages: C #, Java, PHP, Python. After the first versions of these implementations, we will consider further extensions from which we mention:

(1) List of operations: the client requests the available operations (for XML-RPC and SOAP it is clear how to implement this feature, however for REST it is still a challenge)
(2) How should errors be handled?
(3) A long term proposal: specification and implementation of a code generator that is using the wrapper; code will be generated from other web services frameworks.

References

[1] Bean J., *SOA and Web Services Interface Design, Principles, Techniques and Stardards*. Elsevier, 2010.

[2] Boian F. M., *Servicii web; modele, platforme aplicatii*. Ed. Albastra, Cluj, 2010 (to appear).

[3] DaSilva S.D., *REST PHP Classes*. http://diogok.users.phpclasses.org/browse/author/529977.html

[4] Ferrara A., Mac.Donald M., *Programming .NET Web Services*. O'Reilly, 2002.

[5] Homorodean D., Boian F.M., *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.

[6] Jancso B., *RESTful Web Services*. Proceedings "Zilele Academice Clujene 2010 (ZAC2010)", Ed. Presa Universitar Clujean, Cluj 2010, ISSN 2066-5768, pp. 158–163.

[7] Laurent S. St., Johnson J., Dumbill E., *Programming Web Services with XML-RPC*. O'Reilly, 2001.

[8] Ploscar A., *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

[9] Richardson L., Ruby S., *RESTful Web-Services*. O'Reilly, 2007.

[10] Tidwell D., SNall J., Kulchenko P., *Programming Web-Services with SOAP*. O'Reily, 2001.

[11] * * * *apache-cxf-2.2.7*. http://axis.apache.org

[12] * * * *apache-xmlrpc-3.1.3*. http://ws.apache.org/xmlrpc

[13] * * * *CherryPy-3.1.2.win32.exe*. http://www.cherrypy.org

[14] * * * *Developing a REST Web Service using C# - A walkthrough*. http://www.codeproject.com/KB/webservices/RestWebService.aspx

[15] * * * *Restful Python*. http://github.com/jkp

[16] * * * *Programming with NuSOAP*. http://www.scottnichol.com/nusoapprog.htm

[17] * * * *Python 2.65 Documentation*. http://www.python.org

[18] * * * *RESTEasy JAX-RS: RESTFul Web Services for Java*. http://jboss.org/resteasy

[19] * * * *xmlrpc-2.2.2*. http://ws.apache.org/xmlrpc

[20] * * * *XmlRpcCS-1.2*. http://xmlrpccs.sourceforge.net

Babes-Bolyai University, Faculty of Mathematics and Computer Science, 1 M. Kogalniceanu St., 400084-Cluj-Napoca, Romania

E-mail address: florin@cs.ubbcluj.ro, cdsd0192@scs.ubbcluj.ro, cdan@cs.ubbcluj.ro, hdsd0203@scs.ubbcluj.ro, bea@cs.ubbcluj.ro, adina@cs.ubbcluj.ro