STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 3, 2012

UNIFORM SOLUTIONS FOR WEB SERVICES

FLORIAN BOIAN AND BEATA JANCSO

Abstract. Web Services became widely used in today's software environment. Almost every distributed software application needs at least one Web Service to enhance its functionality. A distributed application can be easily created by using the WSWrapper component. WSWrapper offers a unique and uniform solution for Web Service implementation and integration. An important component of WSWrapper called WSGenerator is presented. 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. For case studies a simple Web Service named HugeIndexOfFiles is defined. This service provides interfaces for the major Web Service types: XML-RPC, SOAP and REST. For each variant the service is described using the standards: XRDL, WSDL and WADL.

1. Introduction

The Web Service technology becomes an important component of todays distributed software environment. This technology is so widely used because of its ability to realize platform independent communication between the components of a distributed application. Another important aspect to mention is the fact that a Web Service can return the requested information in several standardized formats (*XML, JSON, HTML*) which can be consumed by different type of clients, such as: browser based clients, rich desktop applications, and other business applications running on smart portable devices.

There are a couple of available solutions for Web Services. The majority of these frameworks is platform dependent, and do not offer a unique and uniform solution for all three major Web Service types: ***RPC [15]***, ***SOAP [12]*** and ***REST [10]***. Because of the lack of a uniform Web Service solution,

Received by the editors: August 17, 2012.

2010 *Mathematics Subject Classification.* 68U35, 68M11, 68N30, 68N25.

1998 *CR Categories and Descriptors.* H.3.5 [**Information Storage and Retrieval**]: Online Information Services – *Web-based services*.

Key words and phrases. Web Service, WSWrapper, WSGenerator, Formal description of interfaces.

a new solution called ***WSWrapper*** was created and presented in the papers [1, 2, 3, 4, and 8].

The ***WSWrapper*** framework offers a unique and uniform solution for creating Web Services for all the major Web Service types: ***XML-RPC***, ***SOAP*** and ***REST***. All in all, the system provides [1, 2] a unique set of objects, regardless of the implementation language or customer service. In order to implement a service, a user must perform the following actions:

- provide a set of classes, functions, or methods that is exported from the service to be executed by the clients
- define an object ***WebService [2]*** passing the service address, name and type (where type can be XML-RPC, SOAP or REST)
- define mappings for the service methods
- define actions to deploy the service

In order to implement a Web Service client, the user must perform the following actions:

- define a ***WebServiceClient [2]*** object passing the service address, name and type (where type can XML-RPC, SOAP or REST)
- invoke the "***call***" method passing the method to call, and the list of parameters

WSWrapper was implemented in the following programming languages: ***Java***, ***Python***, ***PHP*** and ***C #***.

The WSWrapper framework also offers a uniform solution for Web Service proxy generation. This new component it is called ***WSGenerator [5]*** and was added recently to the WSWrapper framework. Having a uniform Web Service proxy generator library is usefully especially for applications which use functionality supplied by several services. Using several libraries for proxy generation is not ideal, and maintaining such an application could become a nightmare. Also creating the proxy from scratch for each of the used Web Services is not the best solution. ***WSGenerator*** was introduced because of the lack of a uniform and platform independent, automated proxy generation library. The intention of ***WSGenerator*** is to simplify the development phase of a distributed application.

2. The WSGenerator component

The proxy generation component was added to the ***WSWrapper*** framework with the intention to create a complete, platform independent, and uniform set of solutions for Web Services. This new component is called ***Web Service Proxy Generator*** or ***WSGenerator***.

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 *[3]* for all the three major Web Service types: RPC, SOAP and REST. By using this component, generating a proxy for a given service becomes an easy process. The Web Service specific details and operations, including specific transformations remain hidden to the end-user.

At the moment WSGenerator provides solutions for generating client proxies in the following popular programming languages: ***Java***, ***C#***, ***Python*** and ***PHP***. In the future, support for other programming languages will be added.

The ***WSGenerator*** component can be used for generating a client proxy for any Web Service, as long as the Web Service has a machine processable service description file published. The service description file format depends on the type of the Web Service. WSGenerator uses the service description file formats shown in Table 1 for proxy generation.

Web Service type	RPC	SOAP	REST
Format	WSDL XRDL	WSDL	WADL WSDL 2.0 HTTP binding extension

TABLE 1. Web Service description file formats

In practice not all the Web Services publish service description files. Especially Web Services of types RPC and REST do not have any description file. As a consequence generating a proxy for RPC and REST services becomes a real challenge. If the service description file is missing automatic proxy generation becomes much more complicated than the manual proxy creation.

Generating a proxy for a given Web Service is a simple process and can be characterized by the following steps:

- ***step1***: the end-user requests a proxy for service X
- ***step2***: ***WSGenerator*** generates a proxy for service X and returns it to the end-user. The end user receives an executable file containing the generated proxy. For example if the end user request the generation of a proxy for a Java service, he will receive a .jar file.

WSGenerator performs the following actions for proxy generation:

- connects to service X
- reads the service description file
- generates the proxy for service X
- returns the generated proxy to the end user

If there was any error during the proxy generation process, a corresponding exception will be returned to the end-user.

At an abstract level the ***WSGenerator*** component can be defined by the interface shown in Figure 1.

FIGURE 1. WSGenerator interface

The domain objects used by the ***WSGenerator*** interface are represented in the diagram from Figure 2.

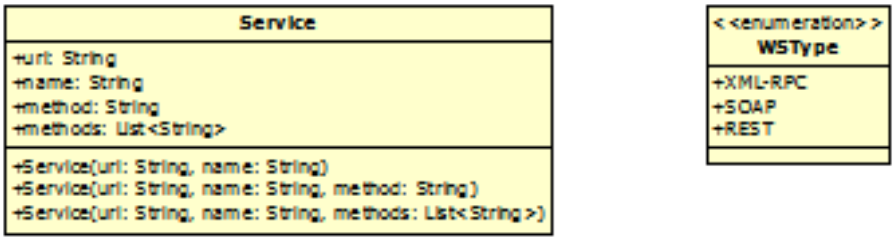

FIGURE 2. WSGenerator domain objects

At the moment the WSGenerator component uses only two domain objects: ***Service*** and ***WSType***. The ***Service*** interface represents at an abstract level a Web Service, and the ***WSType*** represents the type of the Web Service. Currently the following three major Web Service types are supported: ***XML-RPC***, ***SOAP*** and ***REST***. Besides these three Web Service types,

support for other types will be added in the future.

In some cases a Web Service might require some sort of authentication from the client application. Therefore support for authentication will be added to the next version of WsGenerator interface.

Using the ***WSGenerator*** an end-user can also generate a proxy for only one or a few methods of a given Web Service. This feature can be useful when

the client application uses only one or a few methods of a given Web Service. In such cases it makes more sense to include only the necessary methods in the proxy, instead of including all public methods of the service.

3. Case study: HugeIndexOfFiles web services

In order to test the qualities and observe the possible issues of the ***WSWrapper*** framework and ***WSGenerator*** component a web service called HugeIndexOfFiles was created.

The ***HugeIndexOfFiles*** web service indexes in databases some characteristics of the files from various directories, archives, and hosts. The number of indexed files can be very large, until tens of millions.

After the indexing has finished several information can be obtained. As an example we mention the following:

- files whose name get a check pattern,
- duplicated files (possibly with different names)

The indexing database has three tables: head, files and archives, presented in Figure 3.

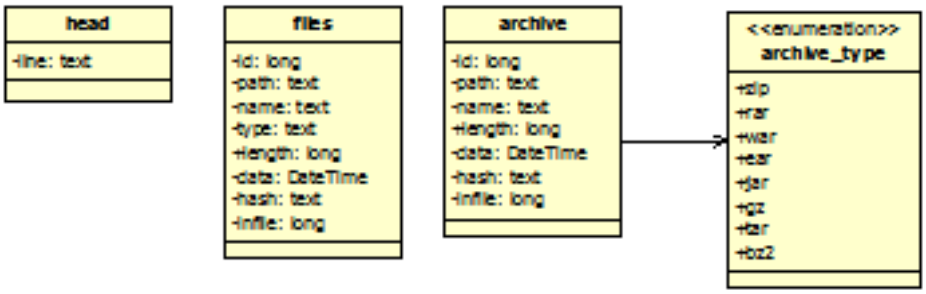

Figure 3. HugeIndexOfFiles indexing database tables

The fields of the indexing database have the following meaning:

- ***line***: line contains a statistical summary of the database content like:

Sat May 19 21:28:09 2012, db: flocopii.db, files: 412784, differentFiles: 221151, filesInArchives: 5961658, hiddenFiles: 10958, roots: +f:/

- ***id***: represents the primary key of the line from the table
- ***path:*** contains the URL of hosting machine and the path in the file system of the file,
- ***name:*** is the name of the file,

- ***type:*** represents the extension of the name (like .c, .pdf, .txt etc.),
- ***length:*** represents the size of the file in bytes,
- ***data:*** represents the creation date and time,
- ***hash***: is a SHA1 hash of the contents of the file, in base64 representation,
- ***infile***: is 0 if the file is not member of any archive file, or is the id of the archive file member. If the file is member of more nested archives, then infile will point to the largest from the archives member.

The public interface of the ***HugeIndexOfFiles*** is illustrated in Figure 4.

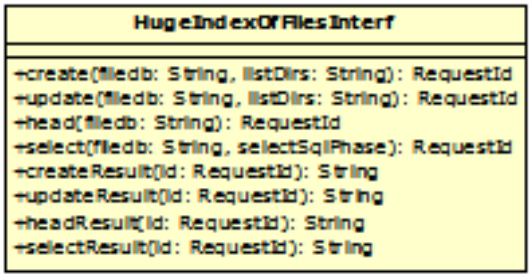

FIGURE 4. HugeIndexOfFiles interface

In some cases the indexing process can take several hours or even more days. As a consequence the ***HugeIndexOfFiles*** has four pairs of methods: one for the request, and one for the response. This policy was adopted in order to obtain the service response asynchronously. In this way, the clients will not be blocked until the service finished the indexing process. The service maintains a pool of requests and a corresponding pool of responses. A request-response pair will be removed only after the client had obtained the final response.

As observed in Figure 4 the ***HugeIndexOfFiles*** interface methods uses the following arguments:

- ***RequestId***: is a long integer that uniquely identifies a request, it is similar to the ***serialVersionUID*** used in the case of distributed applications,
- ***filedb***: contains the URL and absolute name of database,
- ***listDirs***: contains a list of directories and has the form: *DIR1, DIR2,. . . , DIRn.* A directory has the following form: ***{+|0|-} URL Absolute-path.*** The ***+*** prefix is used in order to find files in the archives. The ***0*** prefix is used to ignore the archives, and the – prefix is used to remove files from a directory in the index.

- ***selectSqlPhrase***: contains a valid SQL query

The main pair of methods of the HugeIndexOfFiles is: ***select*** and ***selectResult***. The ***head*** method checks the database consistency. If the database is in an inconsistent state, the ***head*** method transforms it so that the loss to be minimal.

For creating a new database for a list of directories the ***create*** method can be used. If the ***createResult*** or ***updateResult*** methods are invoked before the ***create*** or ***update*** operations had finished, the service response will be something like this:

```
6360000 files handled Sat May 19 21:14:54 2012
f:/Sticuri/BFMntfs2Go/01Compendiu.zip/01Compendiu/Echo.class
```

An example of a final response from ***createResponse*** and ***updateResponse*** is shown in Table 2.

createResponse	***updateResponse***
412784 inserted files. 5961658 inserted files from archives. 0 ignored files from -DIR's. Sat May 19 21:28:09 2012, db: flocopii.db, files: 412784, differentFiles: 221151, filesInArchives: 5961658, hiddenFiles: 10958, roots: +f:/ create stop. Time 613139 seconds.	186354 intact files. 0 intact files from archives. 5 created files. 18 created files from archives. 5 modified files. 7 deleted files. 0 deleted files from archives. 0 ignored files from -DIR's. Sat Apr 28 15:08:06 2012, db: rlfdata.db, files: 186364, differentFiles: 99058, filesInArchives: 3522260, hiddenFiles: 1351, roots: +d:/ update stop. Time 3426 seconds.

TABLE 2. createResponse and updateResponse response example

The main components of the ***HugeIndexOfFiles*** web service are implemented in the Python programming language. In order to test the qualities and defects of the ***WSWrapper*** and ***WSGenerator*** three implementation of the ***HugeIndexOfFiles*** were created using: ***XML-RPC***, ***SOAP*** and ***REST***. Because a ***SOAP*** service will create by default a ***WSDL [13]*** document, in the following paragraphs only the ***XRDL [16]*** and ***WADL [7]*** documents of the ***HugeIndexOfFiles*** web service implementations are presented.

In the case of the ***XML-RPC*** web service an XRDL document is created. The XRDL file of the HugeIndexOfFiles is illustrated in Figure 5.

```
<?xml version="1.0" encoding="UTF-8"?>
<service name="HugeIndexOfFiles XML-RPC service"
    ns="ro.ubbcluj.cs.hugeindexoffiles" url="www.scs.ubbcluj.ro">
    <types>
        <type name="RequestId" type="long" />
    </types>
    <methods>
        <method name="create" result="RequestId">
            <param type="string">filedb</param>
            <param type="string">listDirs</param>
        </method>
        <method name="update" result="RequestId">
            <param type="string">filedb</param>
            <param type="string" mandatory="false">listDirs</param>
        </method>
        <method name="head" result="string">
            <param type="string">filedb</param>
        </method>
        <method name="select" result="string">
            <param type="string">filedb</param>
            <param type="string">selectSqlPhrase</param>
        </method>
        <method name="createResponse" result="string">
            <param type="RequestId">id</param>
        </method>
        <method name="updateResponse" result="string">
            <param type="RequestId">id</param>
        </method>
        <method name="headResponse" result="string">
            <param type="RequestId">id</param>
        </method>
        <method name="selectResponse" result="string">
            <param type="RequestId">id</param>
        </method>
    </methods>
</service>
```

FIGURE 5. HugeIndexOfFiles XRDL document

In the case of the ***RESTful HugeIndexOfFiles*** web service the parameters ***filedb*** and ***id*** will be the last fields from the URI. The methods of the service can be invoked as presented in the table shown in Table 3.

URI	HTTP method	HTTP body
http://address/create/{filedb}	POST	listDirs
http://address/update/{filedb}	PUT	listDirs?
http://address/head/{filedb}	GET	-
http://address/select/{filedb}	PUT	selectSqlPhrase
http://address/createResponse/{id}	DELETE	-
http://address/updateResponse/{id}	DELETE	-
http://address/headResponse/{id}	DELETE	-
http://address/selectResponse/{id}	DELETE	-

TABLE 3. HugeIndexOfFiles RESTful web service method invocation

The operations of the RESTful ***HugeIndexOfFiles*** web service are described using a ***WADL [9]*** document. An excerpt from the ***WADL*** document is presented in Figure 6.

4. Conclusions

The majority of the popular distributed software application uses one or more Web Service supplied functionality. Manual creation of several client proxies is not ideal. Also using several different libraries for automatic proxy generation might not be a good solution.

The ***WSGenerator*** component offers a simple, easy to use, and unique tool for Web client proxy generation. WSGenerator becomes a unique tool because of its uniform interface and support for the most popular Web Service types. The client proxy is created based on the service descriptor file of the Web Service. If the service does not publish any service description file, automatic proxy generation might be difficult or even impossible. This might be an issue especially in the case of ***RPC*** and ***RESTful*** Web Services. As a possible solution to this issue a service descriptor generator component will be added to the WsGenerator tool. In order to preserve the unified interface of WsGenerator the new component will provide solution for all Web Service types. Besides this new component also support for authentication and JSON-RPC [10] will be added in the future.

The ***HugeIndexOfFiles*** web service was created in order to test the ***WSGenerator*** component. In order to create a uniform implementation for the ***HugeIndexOfFiles*** web service the ***WSWrapper*** framework was used. The ***HugeIndexOfFiles*** service can be easily integreated into any application by using the ***WSGenerator*** component.

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
    xmlns:tns="urn:yahoo:yn" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:yn="urn:yahoo:yn" xmlns:ya="urn:yahoo:api" xmlns="http://wadl.dev.java.net/2009/02">
    <grammars>
        <include href="HugeIndexOfFiles.xsd" />
    </grammars>
    <resources base="http:// www.scs.ubbcluj.ro/HugeIndexOfFiles/V1/">
        <resource path="create/{filedb}">
            <method name="POST" id="createid">
                <request>
                    <param name="listDirs" type="xsd:string" style="query"
                        required="true" />
                </request>
                <response status="200">
                    <representation mediaType="application/xml" element="yn:RequestId" />
                </response>
            </method>
        </resource>
        <resource path="update/{filedb}">
            <method name="PUT" id="updateid">
                <request>
                    <param name="listDirs" type="xsd:string" style="query"
                        required="false" />
                </request>
                <response status="200">
                    <representation mediaType="application/xml" element="yn:RequestId" />
                </response>
            </method>
        </resource>
        <resource path="head/{filedb}">
            <method name="GET" id="headid">
                <response status="200">
                    <representation mediaType="application/xml" element="yn:RequestId" />
                </response>
            </method>
        </resource>
        . . . . . .
        <resource path="createResponse/{id}">
            <method name="DELETE" id="createResponseid">
                <response status="200">
                    <representation mediaType="application/xml" element="yn:String" />
                </response>
            </method>
        </resource>
        . . . . . .
    </resources>
</application>
```

FIGURE 6. HugeIndexOfFiles WADL excerpt

REFERENCES

[1] Boian F. M. *Unification of Web Service Technologies,* Proceedings "Zilele Academice Clujene 2010 (ZAC2010)", Ed.Presa Universitara Clujeana,Cluj 2010, ISSN2066-5768, pp. 92-97

[2] Boian F.M. Chinces D, Ciupeiu D, Homorodean D, Jancso B, Ploscar A. *WSWrapper – A Universal Web service generator,* Studia Univ. Babes-Bolyai, Volume LV, Number 4, 2010, pp. 59-69
[3] Boian F.M. *Servicii web; modele, platforme, aplicaii.* Ed. Albastr, Cluj, 2011, pp. 363-369
[4] Boian F.M. *An uniform approach to define and implement the web services; case studies for indexing huge file systems*, Proceedings "Zilele Academice Clujene 2012 (ZAC2012)", pp.1-6
[5] Jancso B. *RESTful Web Services,* Proceedings "Zilele Academice Clujene 2010 (ZAC2010)", Ed.Presa Universitara Clujeana, Cluj 2010, ISSN2066-5768, pp.158-161
[6] Jancso B. *Web Service proxy Generator,* Proceedings "Zilele Academice Clujene 2012 (ZAC2012)", pp.1-6
[7] Marc J. Hadley - Web Application Description Language, 2006
[8] Ploscar A. A *Java Implementation for REST-style web service* ,Proceedings "Zilele Academice Clujene 2010 (ZAC2010)", Ed.Presa Universitara Clujeana, Cluj 2010, ISSN2066-5768,ZAC2010, pp.140-146
[9] Takase T. Makino S. Kawanaka S. Ueno C.F. Ryman A. *Definition Languages for RESTful Web Services: WADL vs. WSDL 2.0.* http://www.ibm.com/developerworks/library/specification/ws-wadlwsdl/index.html
[10] *** JSON-RPC, http://json-rpc.org
[11] *** REST principles, http://www.ics.uci.edu/∼fielding/pubs/dissertation/rest_arch_style.htm
[12] *** SOAP, http://www.w3.org/TR/soap
[13] *** WSDL, http://www.w3.org/TR/wsdl
[14] *** WSDL2.0 HTTP Binding Extension, http://www.w3.org/TR/wsdl20-adjuncts/#http-binding
[15] XML-RPC, http://xmlrpc.scripting.com/spec.html
[16] XRDL, http://code.google.com/p/xrdl

Babeş-Bolyai University, Department of Computer Science, 1 M. Kogălniceanu Street, 400084 Cluj-Napoca, Romania
E-mail address: florin@cs.ubbcluj.ro
E-mail address: bea.jancso@yahoo.com