Web Service to JSON-RPC Transformation

Christian Samsel, Sevket Gökay, Paul Heiniz and Karl-Heinz Krempels
RWTH Aachen University, Informatik 5, Ahornstr. 55, 52074 Aachen, Germany

Keywords: Web Services, XML, SOAP, WSDL, JSON, JSON-RPC, Transformation, Mobile Development.

Abstract: During the last years JavaScript Object Notation Remote Procedure Call (JSON-RPC) emerged as the de facto standard for service calling on mobile devices. Unfortunately many enterprise services are still only available as traditional Web Service accessible via Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP). In this paper we introduce the Web Service to JSON-RPC adapter tool which offers JSON-RPC ports matching the SOAP ports in a WSDL-based web service definition. The adapter automatically translates JSON-RPC requests incoming on these ports to a responding SOAP message and forwards it the SOAP server. The SOAP response is translated back to JSON-RPC and delivered to the original client. Our adapter enables software developers to use a JSON-RPC client which is well supported on mobile platforms to access SOAP-based Web Services without altering the server nor requiring additional software on client side.

1 INTRODUCTION

Since the Internet gains more and more importance in our lives there is an explosive growth of information in the world. To cope with such big data, an environment of automated and interactive systems is needed. This causes a shift from an Internet where humans were the only actors who request information to the Internet of things, where applications and devices can also communicate with each other.

Such application communication services are realized between a client (*sevice requester*) and a server (*service provider*). The interaction involves the client sending a request message with some parameters to the server to execute a procedure and the server sending a response (return values) back to the client. This is called a Remote Procedure Call (RPC).

> A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.[1]

In this work we stick to this definition of Web Service. The later described JSON-RPC is explicitly exclude from the definition of Web Service for an easier distinction.

An alternative way of realizing the communication between a client and a server is JavaScript Object Notation Remote Procedure Call (JSON-RPC) (JSON-RPC Working Group, 2012). In JSON-RPC the messages are encoded in JSON which is a data representation, serialization and interchange format with similar purpose as Extensible Markup Language (XML). JSON is part of the JavaScript standard but still language independent. JavaScript Object Notation (JSON) is very simple and therefore easy to generate, to parse, and has only a small memory footprint.

We deliberately do not use the term *RESTful* in this paper to not mix up Representational State Transfer (REST) and JSON-RPC. REST is a collection of paradigms whereas JSON is a technology which eventually can fulfill REST. Our tool purely works on a technology level and does not deal with paradigms.

1.1 Motivation

Web Services induce a massive overhead in terms of file size and parse effort mainly due to the use of XML

[1]Definition Web Service in http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/

Samsel C., Gökay S., Heiniz P. and Krempels K..
Web Service to JSON-RPC Transformation.
DOI: 10.5220/0004499502140219
In *Proceedings of the 8th International Joint Conference on Software Technologies* (ICSOFT-EA-2013), pages 214-219
ISBN: 978-989-8565-68-6

as serialization format (Lawrence, 2004; Cook and Barfield, 2006; Pautasso et al., 2008; Nurseitov and Paulson, 2009; Sumaray and Makki, 2012). For example, compare the size of the JSON-RPC and Simple Object Access Protocol (SOAP) requests in 1 and 2. Both essentially contain the same payload. For stationary computers and servers these drawbacks are negligible because of the usual high internet connection speed and high processing powers for these devices. But in todays Internet the percentage of mobile and low power devices like smartphones, tablets, embedded systems (e.g. in cars), etc is high. RPC using Web Services is not desirable for such devices. Instead, JSON is emerging as new standard for mobile service calling. Frameworks for JSON are a built in part of every noteworthy mobile platform development environment.

Additionally, many existing enterprise solutions are solely available through Web Services, but these are not well supported by mobile development kits. Unfortunately, both Web Services and JSON-RPC are not interoperable. We propose a solution to this issue.

The remainder of this paper is structured as follows: In 2 existing approaches and the used tools are discussed. 3 describes our approach to the problem on an abstract level, whereas 4 introduced our prototype implementation. In 5 an evaluation is proposed and 6 concludes the work.

2 RELATED WORK

This section lists recent related research and the used tools.

(Wang, 2011) describes the process of translating between XML and JSON for web applications. Unfortunately, no proof-of-concept nor implementation details are given. Instead of operating on real Web Services and real data encoded in SOAP and JSON-RPC, it is only vaguely spoken of XML and JSON. It is questionable if the approach can be applied to real world applications and if the evaluation results are valid.

2.1 Tools

Apache Camel[2] is a framework for enterprise application integration. Its main part is a routing-engine builder which is used to define routing rules, data sources and destinations. It also supports protocol conversions. It is a mature open source project. Apache Camel is the basis for our tool (Ibsen et al., 2011).

StAXON[3] allows to read and write JSON using the Java Streaming API for XML, thus it is a convenient tools for syntactic translation between JSON and XML.

All presented tools are available under the Apache Software License, Version 2.0[4].

3 APPROACH

This section describes our approach on an abstract level from the view of the developer using our tool. Assume the following scenario: A developer intends to create a mobile application using a specific existing Web Service. This Web Service is traditionally SOAP / Web Services Description Language (WSDL)-based and cannot be changed. Unfortunately, most mobile frameworks only support JSON / JSON-RPC for service calling.

3.1 Workflow

We opted for a two-stage approach to this problem. A few preparation steps are required to shove the tool between the Web Service (WS) server and JSON-RPC client. The developer has to login into the *manager* (see 4.1) configuration interface and enter the URL to the WSDL which describes the Web Service of interest. After the processing is done, the developer can choose to start the *adapter* (see 4.3) or if preferred download the archive instead (e.g. to deploy it elsewhere). From now on no additional runtime configuration is required. The adapter presents an informal description of the JSON-RPC interface on a status web page, which then can be used as a guide to implement the JSON-RPC client. The description contains information about the base URL, the available function names, their expected parameters, and corresponding return values. The GUI utilities JSON Editor Online[5] for user interaction and presentation. It also contains a JSON-RPC simulator for debugging purposes.

3.2 Translation

For translation between XML and JSON we use StAXON (see 2.1) which employs the BadgerFish convention scheme. It consists of nine simple conversions which are powerful enough to project the payload of XML to JSON and vice versa. Here you can see an example rule:

[2]https://camel.apache.org
[3]https://github.com/beckchr/staxon
[4]https://www.apache.org/licenses/LICENSE-2.0
[5]https://github.com/josdejong/jsoneditoronline

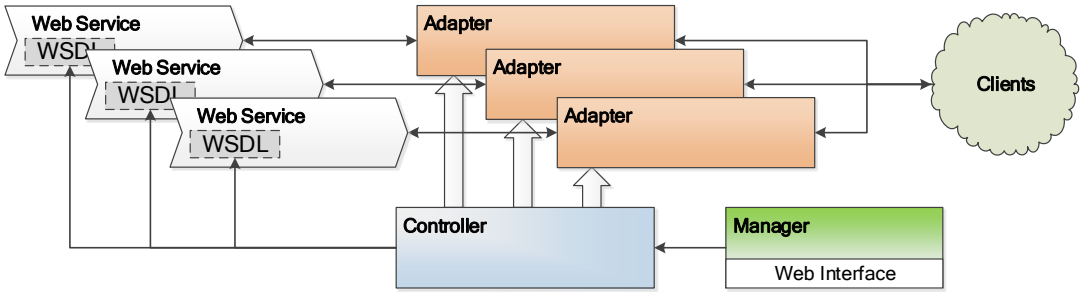

Figure 1: System Architecture. Using manager and controller the user creates and controls independent adapter instances which are responsible for the actual translation between Web Service and JSON-RPCs Clients.

Rule 2: Text content of elements goes in the $ property of an object.
`<alice>bob</alice>`
becomes
`{ "alice": { "$" : "bob" } }`[6]

Using these approaches the developer can conveniently use JSON-RPC without changing the existing Web Service and also profit from its advantages for mobile development (e.g. lightweight memory footprint).

4 IMPLEMENTATION

The Web Service to JSON-RPC translator is implemented as a group of Java servlets. It is designed to run under the Apache Tomcat Java application server. The translator consists of the manager and controller combined in one Java servlet (see 4.1 and 4.2) and one or more adapters (4.3) each as independent servlet. The general architecture is presented in 1.

4.1 Manager

The developer interacts with the frontend component, called manager. The manager is a lean web interface similar to the standard Apache Tomcat servlet manager. The developer can access the manager using username and password and then add, start, stop and delete adapters. To add an adapter the developer enters the URL of the respective WSDL file into the web interface. The manager will then download and analyze the WSDL file and generate a standalone adapter in form of a Java servlet. This Java servlet can either be directly deployed to the local Apache Tomcat instance or downloaded as a WAR archive for deployment on a different host. Locally deployed adapters can also be started, stopped, removed or exported via the manager.

4.2 Controller

The controller is the backend component. It accepts commands received from the manager and is responsible for controlling the independent adapters, e.g., starting and stopping adapter servlets.

4.3 Adapters

2 presents the implementation of the adapter. Adapters are independent servlets specific per WSDL file and therefore per Web Service. The adapters are responsible for the actual translation between JSON-RPC and SOAP. The JSON-RPC client sends a JSON-RPC request to the adapter. The request is parsed and unmarshalled to plain JSON. Using StAXON (2.1) the JSON data is transformed to a XML tree and then embedded into a standard compliant SOAP message and send to the target Web Service. StAXON implements a slightly customized BadgerFish (3.2) scheme. On the reverse path the corresponding SOAP response is unmarshalled to plain XML and converted to JSON using StAXON. After embedding the JSON into JSON-RPC the response is delivered to the original Client.

Hereafter, we present a complete Web Service to JSON-RPC transformation session using the `listAll` function call from the Study Web Service present OpenClinica. It returns all clinical trials listed in OpenClinica to which the authenticated users has access. OpenClinica[7] is a prominent open source application for conduction clinical trials. It contains multiple well documented Web Services with different degree of complexity.

4.3.1 JSON-RPC to SOAP Transformation

We start with a simple JSON-RPC request as it has been generated by a mobile client in 1. The JSON-

[6]http://www.sklar.com/badgerfish/
[7]https://www.openclinica.com

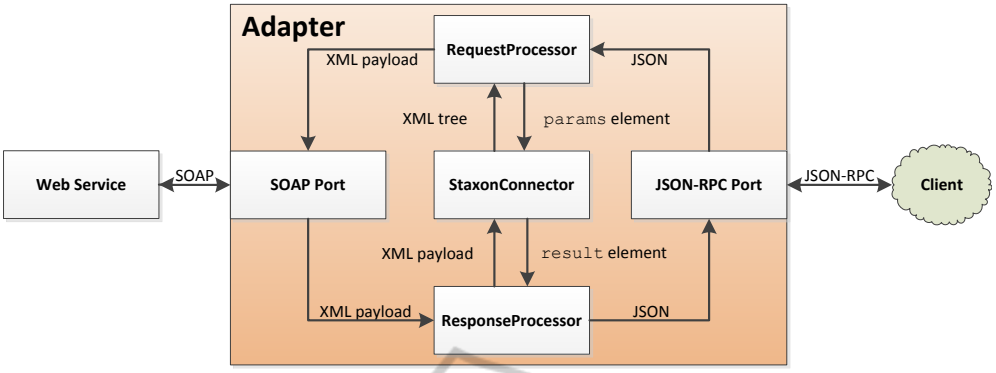

Figure 2: Adapter Implementation. The adapter receives a JSON-RPC request on its JSON-RPC port which is then translated to a analogous SOAP message and sent to the respective SOAP port. The accruing SOAP response is parsed and translated back to JSON-RPC and delivered to the original client.

RPC field `id` is set by the client to an arbitrary alphanumeric value (usually just integers) and carried over in the response so the client can correlate unordered responses to their respective requests. The `SOAP-HEADER` is transformed to a `XwsSecurity`[8] header for authentication as expected by the OpenClinica Web Service.

```
1 {
2   "jsonrpc": "2.0",
3   "id": 1337,
4   "method": "v1:listAllRequest"
5   "params":
6   [
7     "SOAP-HEADER":
8       ["username": root,
9        "password": 123456]
10  ]
11 }
```

Listing 1: JSON-RPC Request. A minimal JSON-RPC call only containing the required elements `jsonrpc` and `id` plus the authentication credentials and the method name.

The `id` is not used in the SOAP request but instead handled by an internal state in the adapter. The SOAP request shown in 2 is enriched with XML namespace information extracted from the WSDL file.

```
1 <?xml version="1.0"?>
2
3 <soapenv:Envelope
4 xmlns:soapenv="http://schemas....oap/envelope/"
5 xmlns:v1="http://openclinica.org/ws/study/v1">
6
7 <soapenv:Header>
8 <wsse:Security soapenv:mustUnderstand="1"
9 xmlns:wsse="http://docs.oas...secext-1.0.xsd">
10 <wsse:UsernameToken wsu:Id="UsernameToken-..."
11 xmlns:wsu="http://docs.oas....tility-1.0.xsd">
12 <wsse:Username>root</wsse:Username>
13 <wsse:Password
14 Type="http://docs.o....ofile-1.0#PasswordText">
15 password</wsse:Password>
16 </wsse:UsernameToken>
17 </wsse:Security>
18 </soapenv:Header>
19
20 <soapenv:Body>
21 <v1:listAllRequest/>
22 </soapenv:Body>
23
24 </soapenv:Envelope>
```

Listing 2: SOAP request (shortened). A minimal SOAP call only containing the required SOAP elements; authentication credentials and the method name. Namespace informations have been trimmed.

4.3.2 SOAP to JSON-RPC Transformation

We start with the response created by the OpenClinica Web Service in 3. It contains a result status field with the contents *success* and the requested payload object `studies`. The object `studies` contains one object of complex type `study` containing information about the respecting study like the unique identifier.

The translated response is shown in 4. It contains the result distilled from XML to JSON plus the required JSON-RPC fields. It is valid JSON-RPC and can be parsed easily by common JSON frameworks. Complex types like `study` in 3 are converted to a representation of Arrays and Strings.

4.3.3 Security

For confidential transmission the client to adapter connection can be SSL-secured using the the standard Apache Tomcat facilities. For example an externally signed certificate can be used. The adapter to Web

[8]https://static.springsource.org/spring-ws/sites/1.5/reference/html/security.html

```
1
2 <?xml version="1.0"?>
3
4 <SOAP-ENV:Envelope
5 xmlns:SOAP-ENV="http://schemas...oap/envelope/">
6 <SOAP-ENV:Header/>
7
8 <SOAP-ENV:Body>
9 <listAllResponse
10 xmlns="http://openclinica.org/ws/study/v1">
11 <result>Success</result>
12 <studies>
13 <study>
14 <identifier>default-study</identifier>
15 <oid>S_DEFAULTS1</oid>
16 <name>Default Study</name>
17 </study>
18 </studies>
19 </listAllResponse>
20 </SOAP-ENV:Body>
21
22 </SOAP-ENV:Envelope>
```

Listing 3: SOAP response (shortened). A minimal SOAP response only containing the required SOAP elements; authentication credentials and the result array.

Service connection is used as noted in the WSDL[9], either with SSL or in plaintext. If SSL is used, a certificate check can be optionally forced. The manager is secured by default using a username / password authentication to prevent abuse. If required, the adapters can also be protected to restrict access.

4.3.4 Availability

The adapter has no special failover or other redundancy features incorporated. Instead the idea for a reliable function is to deploy the same adapter multiple times to different locations and use them in a round robin fashion. Alternatively a load balancer can be used to distribute incoming requests to different adapter instances. Although, more components might increase the aggregate delay to a point the user experience suffers.

```
1 \vspace{-1cm}
2 {
3   "jsonrpc": "2.0",
4   "id": 1337,
5   "result": {
6     { "result" : success }
7     { "studies" :
8       [ "study" :
9         {"identifier": "default-study",
10        "oid": "S_DEFAULTS1",
11        "name": "Default Study"}
12      ]
13    }
14  }
15 }
```

Listing 4: JSON-RPC response. A minimal JSON-RPC response only containing the required elements `jsonrpc` and `id` plus the requested playload containing the study name.

4.3.5 Error Handling

The adapter itself only does syntax checking on incoming transmissions. Neither JSON-RPC nor SOAP messsages are checked for conformity to e.g. the respective schema. Instead we rely on the checks on the endpoints and just pass-through potential error messages using the approriate measures for JSON-RPC (`error` object) and SOAP (`Fault` element). This also applies to miscellaneous error message, for example, in case of an overloaded server.

5 EVALUATION

Proof of functionality of our tool and the approach to a performance evaluation of our tool are presented in this chapter.

5.1 Functionality

For functionality demonstration we use an existing test installation of OpenClinica, an Open Source application for clinical research. Excerpts of a real session can be found in 4.3. Additionally, we tested Web Services available online, including but not limited to entries in 1.

5.2 Performance

Performance measurements are currently a subject of work in progress. We will evaluate the overhead induced by the adapter in both terms of additional delay and processing overhead. To calculate the overhead delay we will time simulated JSON-RPC requests using the adapter to a test Web Service and compare the results to equivalent direct SOAP requests. For efficiency measurement we will add the adapter to the host running the Web Service and compare served requests per second for equivalent requests in both JSON-RPC and SOAP. As the setup processing only occurs once, we consider its effort negligible.

[9]Depending whether URL starts with https:// or http://

Table 1: List of tested public Web Services.

Name	Function	WSDL Uniform Resource Locator (URL)
BLZService	Get bank name by BLZ (german identification number for banks)	`http://www.thomas-bayer.com/axis2/services/BLZService?wsdl`
ConvertSpeeds	ConvertSpeed units (e.g. mph to kmh)	`http://www.webservicex.net/ConvertSpeed.asmx?WSDL`
Weather	Provides weather information in USA by zip code	`http://wsf.cdyne.com/WeatherWS/Weather.asmx?WSDL`
Calculator	Performs arithmetic operations	`http://soaptest.parasoft.com/calculator.wsdl`

6 CONCLUSIONS AND FUTURE WORK

We presented a tool which translates JSON-RPC requests to SOAP requests and SOAP responses to JSON-RPC responses based on information present in WSDL. This approach enables mobile JSON-RPC clients to access traditional Web Services without altering the Web Service application and only requiring JSON-RPC support on client side (instead of the more complex SOAP support).

6.1 Future Work

We are currently working on a mechanism to deliver functioning client stub classes for the iOS and Android plattforms in addition to the informal description of the generated interface. This will simplify mobile development even further.

The architecture of our tool leaves open space to support more RPC standards. We are considering adding support for additional protocols on both client and server sides.

REFERENCES

Cook, W. and Barfield, J. (2006). Web Services versus Distributed Objects: A Case Study of Performance and Interface Design. *IEEE International Conference on Web Services (ICWS'06)*, pages 419–426.

Ibsen, C., Anstey, J., and Zbarcea, H. (2011). *Camel in Action*. Manning.

JSON-RPC Working Group (2012). JSON-RPC 2.0 Specification.

Lawrence, R. (2004). The space efficiency of XML. *Information and Software Technology*, 46(11):753–759.

Nurseitov, N. and Paulson, M. (2009). Comparison of JSON and XML data interchange formats: A case study. In *22nd International Conference on Computers and their Applications in Industry and Engineering (ISCA'06)*.

Pautasso, C., Zimmermann, O., and Leymann, F. (2008). Restful web services vs. big'web services: making the right architectural decision. *17th International Conference on World Wide Web (WWW'08)*, pages 805–814.

Sumaray, A. and Makki, S. (2012). A comparison of Data Serialization Formats for Optimal Efficiency on a Mobile Platform. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication (ICUIMC'12)*.

Wang, G. (2011). Improving Data Transmission in Web Applications via the Translation between XML and JSON. *3rd International Conference on Communications and Mobile Computing (CMC'11)*, pages 182–185.