Analysis of developers choices for API Rest or Soap protocols

Alen Šimec, PhD; Polytechnic of Zagreb; alen@tvz.hr
Lidija Tepeš Golubić, PhD; Polytechnic of Zagreb; Lidija Tepeš Golubić ltepes2@tvz.hr

Abstract - This paper is about analysis and comparison of SOAP protocol and REST architecture, their possibilities and use in everyday scenarios. The structure, way of work and area of application, as well as the most famous cases of application of one or the other approach have been described in short.

Keywords - rest protocol, soap protocol, comparison, applications, World Wide Web, XML language

I. INTRODUCTION

The World Wide Web is the most popular distributed application in history, and Web services and mashups have turned it into a powerful distributed computing platform [2].

By sudden development of computer technology broader world population have gained access to computers and network services. The biggest network in the world, World Wide Web, has become a daily used server for information access and exchange. As the Internet has been growing, the number of servers and services that users access on a daily basis has been growing as well. Nowadays, an average user wants a fast, quality, functional and simple access to information and services. Web service and Internet website designers need to adjust to those demands and from possible available technologies get the maximum needed for fulfilling the expectations of end users.

Considering that all the Internet traffic is based on HTTP protocol, designers of websites, applications and online services have at hand a whole range of different protocols and technologies for creating a service that will make the use of the Internet simpler and more functional. Special attention needs to be paid to the Internet of things, which is a phenomenon of connecting more and more devices through the Internet, and that demands an even higher data flow and more and more ways of connecting those devices with different applications, services and operating systems. The difference of technologies, protocols and program languages that are being used today is making simple communication, mutual connection and information circulation more difficult, and there is a need for some kind of tool that will enable the solving of this problem.

The two most popular tools for independent connecting and communication between applications and services nowadays are SOAP protocol and REST architecture. Those are technologies that have many characteristics in common, but at the same time, many differences as well. What they have in common is giving a possibility of connecting and joint work of different subjects, such as computers, servers, mobile phones, clients, programs, applications and services, and they can all work on different protocols, operating systems and they can be designed by different program languages. SOAP and REST use the application HTTP protocol for transmission of hypertext and messages in XML language.

We cannot say which of those two tools is better because the answer to that question depends on the allocation of service and application that is being used. In some cases SOAP protocol is better, while in the other ones it is preferable to use REST architecture. This term paper will make the analysis of both tools, their structure, through studying their use in the real world and of some of their advantages and disadvantages.

II. SOAP PROTOCOL

SOAP protocol (*Simple Object Access Protocol*) represents a simple and data light mechanism for exchanging structural information between different users in distributed and decentralized surroundings using XML language. Such exchange of information pertains to the implementation of Web service on computer networks. The purpose of SOAP protocol is to encourage extensibility, neutrality and independence. This pertains to continuous development of extensions related to safety and WS-routing, possibility of working on any kind of protocol (e.g. HTTP, SMTP, TCP, UDP or JMS), and the possibility of using any kind of programming model. Considering that it uses XML (Extensible Markup Language) for forming messages, it depends on protocols of transport layer, such as HTTP (*Hypertext Transfer Protocol*) and SMTP (*Simple Mail Transfer Protocol*) for exchange and transmission of messages.

SOAP enables mutual communication through XML for the processes that are being started on different computers and different operating systems. Considering that HTTP protocol has been installed on all the operating systems, the clients can access Web services and receive responds from them regardless the language or the platform. This chapter will be about a short review of SOAP protocol history, messages structure, way of work and its application.

SOAP was designed and mentioned for the first time as a protocol for accessing objects in 1998. Its authors are Dave Winer, Don Box and Bob Atkinson and Moshen Al-Ghosein who worked for Microsoft at the time. Due to Microsoft's business policy of postponing the publishing, the

specification was officially published only at the end of 1999. Dave Winer published a part of XML-RPC specification in 1998 by himself because of Microsoft's hesitation. The first version of SOAP did not use XML, but primitive types of data that were being written into structure and sent and received by specifically defined operations and methods. The second development phase lasted from 1999 until May 8 2000 when the first version 1.1 was given to W3C with IBM as a co-author.

This version was much more extensible which made the use of explicitly Microsoft technologies unnecessary. The fact that immediately after that IBM published a Java SOAP implementation and donated it to the Apache XML Project for development of the open code, has convinced even the strongest skeptics that SOAP is something that deserves attentions. Whence the Sun expressed their support to SOAP protocol, it did not take a long time for other manufacturers to join and start working on the implementation of Web service. During the year 2000 W3C formed a working group that started to design a basic XML protocol that was supposed to become the core for XML-based distributed computing. They started working on SOAP 1.1 version as a base and the published the first minor version SOAP 1.2 on June 9 2001. This made the XML way of writing data a standard and SOAP attracted attention of all major companies that continued its further development after which the protocol became a recommendation for building client applications.

Specification of SOAP protocol describes a structure that can be narrowed to four basic parts:

- SOAP envelope or messages,
- SOAP encoding rules,
- SOAP RPC (*Remote Procedure Call*) presentation,
- Ways of connecting and sending messages.

The envelope defines the framework in which the content of the message is being expressed, as well as for whom it is intended and whether it is optional or mandatory. Encoding rules define the mechanisms that are being used for exchanging data type instances. SOAP RPC presentation defines the convention that may be used for presentation of distant procedures and responses

SOAP message is an XML document that consists of a mandatory SOAP envelope, optional SOAP header, mandatory SOAP body and optional fault SOAP envelope is the highest element of the XML document that represents the message and that indicates the beginning and the end of the message, in order for the recipient to know that they have received the whole message. So, SOAP envelope solves the problem of differentiating between receiving a message and processing a message, and it actually represents a mechanism of packing a message. The envelope is a mandatory part of each message that consists of a header and a body. The header is not a mandatory element, while the body is, and each envelope may contain only one *body* element. Different versions of SOAP protocols have different envelopes and they cannot read the envelopes of the other protocol version. The envelope is being defined through the ENV prefix and the *Envelope* element.

The optional header element gives a flexible frame for defining additional application requests. For example, digital signatures for services that are password protected, number or accounts for which certain content is intended, public encoding keys, etc. can be defined in the header. It is possible to use multiple header elements that are being written as header blocks. SOAP header can have two different attributes: actor attributes and *MustUnerstand* attribute. Since the SOAP protocol defines the message path as a set of service junction points of which each can process and forward messages, the actor attributes determine who exactly is the header recipient. The *MustUnerstand* attribute determines whether the header element is optional or mandatory. If the attribute is set to *true*, the recipient must understand and process the header attribute, otherwise there is an error.

The SOAP message body is a mandatory element that contains XML data that some application will exchange through SOAP messages. The body of the message needs to be contained in the envelope and it needs to follow the header if it has been defined for the message. The message body contains mandatory information that is intended for the end message recipient.

If there is an error while processing a message, the response to the SOAP message is the SOAP element fault that is being returned to the sender in the body of SOAP message. The fault element sends back specific information about the fault, including the predefined code, description and address of SOAP processor that has generated the fault. SOAP message can contain only one fault block.

SOAP protocol consists of a set of inbuilt rules for encoding data types. It enables the SOAP message to mark specific data types, such as *string*, *boolean*, *integer*, *float*, *double* or *array* data types. Data types that are differentiated by SOAP are:

- Scalar types that contain exactly one value, such as the product name, price or description,
- Complex types that contain multiple values, such as orders or a listing of stock inventories.

Scalar data types inherit all the inbuilt simple data types that have been defined by XML specification.

Table 1: Scalar types of SOAP messages

Simple data types built into the XML Scheme	
Simple types	**Data sample**
string	Some text
boolean	true, false, 1, 0.

Float	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
double	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
decimal	-1.23, 0, 123.4, 1000.00
binary	100010
integer	-126789, -1, 0, 1, 126789
nonPositiveInteger	-126789, -1, 0.
negativeInteger	-126789, -1
long	-1, 12678967543233
int	-1, 126789675
short	-1, 12678
byte	-1, 126
nonNegativeInteger	0, 1, 126789
unsignedLong	0, 12678967543233
unsignedInt	0, 1267896754
unsignedShort	0, 12678
unsignedByte	0, 126
positiveInteger	1, 126789.
date	2017-01-15
time	13:20:00.000, 13:20:00.000-05:00.

Processing of SOAP message includes separation of the envelope and processing of information it transmits. SOAP contains a general frame for processing this information, but it leaves the details to the application that uses it. The very idea of a SOAP message is a one-way sending of the envelope from the sender to the recipient, while that envelope may pass through a bigger number of different processors that are between them. SOAP proxy is a web service that might add value or functionality to the transaction between those two. The set of those proxies through which a message passes is called a message path, while each proxy is called an actor. SOAP protocol does not cover a definition of creating the message path, but it defines mechanisms that determine which parts of the message are intended for which actors. Those mechanisms are called "*targeting"* and they are used in relation with message headers. Header blocks may contain attributes that are identifiers of certain actors, and those can be URLs on which there is the wanted actor. After the message is sent, all the proxies that do not correspond to the attribute in the message need to ignore the header block until the message has arrived to the targeted actor. For example, this is the way to use a digital signature to establish secure and valid communication.

There are different extensions, such as Microsoft *SOAP Routing protocol* (*WS-Routing*), that are used for creating the message path. This protocol defines the standard SOAP header block that contains information for routing and defines the exact chain of proxies through which the message must cross.

SOAP protocol is also being used for communication with RPC web services. RPC methods and their parameters are being shown as structures in which the names and the physical order of parameters are being arranged in a way to correspond to the names and order of parameters of the method that is being requested. A similar way is used to form the answers to the method requests.

SOAP protocol that is being used as a standardized protocol for packing that is set up on the network and transport level, does not require one specific transport protocol for message exchange, which makes it extremely flexible to use. SOAP can be transmitted through HTTP, SMTP, FTP, TCP, UDP, JMS, MQSeries or MSMQ protocol.

Due to the Internet's daily routine, the most used protocol for exchange of SOAP messages is HTTP protocol. SOAP specification even describes in detail how the model of exchange of SOAP messages is being mirrored on HTTP. The connection of HTTP and SOAP is also visible in SOAP RPC conventions that consist of a request and of a response, just like HTTP protocol is based on request-response model.

Example [3] of using SOAP protocol through HTTP-a:

REQUEST:

```
POST /StockQuote HTTP/1.1
Content-Type: text/xml
Content-Length: nnnn
SOAPAction: "urn:StockQuote#GetQuote"
<s:Envelope xmlns:s="http://www.w3.org/2001/06/soap-envelope">
...
</s:Envelope>
```

RESPONSE:

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn
<s:Envelope xmlns:s="http://www.w3.org/2001/06/soap-envelope">
...
</s:Envelope>
```

REST is an abbreviation for *Representational State Transfer*. That is an architectural style for designing network applications. The idea of REST is to replace complex mechanisms such as CORBA, RPC or SOAP with HTTP when establishing a communication session between the computer and the service. It can be said that the whole World Wide Web that is based on HTTP is a kind of architecture based on REST. Applications that use REST architecture are called RESTful applications and they use HTTP for data sending, data reading and data deleting (all CRUD operations). REST is not a standard, it does not have a W3C recommendation, it is only a simple way to use the frames of application programming.

REST has been defined by Roy Thomas Fielding in 2000. He defined it in his dissertation "Architectural Styles and the Design of Network-based Software Architectures". He has been developing REST in parallel with HTTP 1.1 protocol and he based it on the 1.0 version from 1996.

Similar to SOAP protocol, REST is also independent from the platform on which it is being developed and used, it is independent from the program language, it is being used through HTTP and it can be easily used through fire-walls.

The key components of REST architecture are resources that are being identified through logic URLs (*Uniform Resource Locator).* Status and functionalities are being presented using resources. Resources are the key element of the real RESTful design such as the methods and services at RPC and SOAP web service. `getProductName` and `getProductPrice` RPC request will not be used, but the product data will be looked at as a resource that contains all the necessary information or connections to the information. Resources can be entities, collections, values or anything else for what a designer might find useful to possess their proper URL.

The resource network is important because it represents the connection of all the resources through mutual links because none of the resources should be too big or contain too many details. Whenever it is possible, a resource should contain links towards additional information, like it is on the web.

The system is based on client-server principle, but the server of one component can be a client of another component.

There is no state of links, interaction is such that the request is independent from the response (stateless). Each new request must contain all the necessary information for a successful execution of such request and it should not depend on prior interactions with that same client.

Resources need to be saved into cache whenever possible. The protocol must enable the server to specify the resources that might be cached and for how long. Headers with control of saving inside of HTTP are being used for this, and clients must respect the server's specifications for saving each used resource.

Proxy servers can be used only as a part of architecture for improving performances and scalability, and any HTTP proxy can be used.

This chapter is going to show on examples how REST is actually simple and "light" when talking about data flow. When we say "light", we mean the quantity of network resources that are used during information exchange.

Example of SOAP request:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap=http://www.w3.org/2001/12/soap-envelope
soap:encodingStyle="http://www.w3.org/2001/12/soap-
encoding">
<soap:body pb="http://www.acme.com/phonebook">
<pb:GetUserDetails>
<pb:UserID>12345</pb:UserID>
</pb:GetUserDetails>
</soap:Body>
</soap:Envelope>
```

The response to this request would be an XML document that is contained inside of the envelope of SOAP response.

Example of REST request:

http://www.acme.com/phonebook/**UserDetails**/**12345**

This is just a URL that would be inside the request body, which is being sent through a simple GET method, and HTTP responds with clear data that are not built inside of any kind of proxy. The data can usually be reached with a HTTP GET method, they can be deleted using POST, PUT or DELETE methods.

These examples show how REST uses nouns, while SOAP uses verbs in order to reach the resources (UserDetails / GetUserDetails).

The server's response in REST is most commonly in an XML file like in the example[2]:

```
<parts-list>
 <part id="3322">
  <name>ACME Boomerang</name>
  <desc>
   Used by Coyote in <i>Zoom at the Top</i>, 1962
  </desc>
  <price currency="usd" quantity="1">17.32</price>
  <uri>http://www.acme.com/parts/3322</uri>
 </part>
 <part id="783">
  <name>ACME Dehydrated Boulders</name>
  <desc>
   Used by Coyote in <i>Scrambled Aches</i>, 1957
  </desc>
  <price currency="usd" quantity="pack">19.95</price>
  <uri>http://www.acme.com/parts/783</uri>
 </part>
</parts-list>
```

During the past few years there has been a large increase in the number of web services which has not diminished the popularity of using SOAP protocol. But nevertheless, the architects of Internet applications are finding reasons to quit SOAP more and more often in favor of a better method for creating web services, which is REST.

REST is not a new technology, it has been present for quite a while as a concept, but it has been realized through technology only recently. While SOAP represents a new phase in Internet development with a set of new specifications, REST shows how the existing principles and protocols are sufficient for creating robust web services and applications because it does not require the use of additional development tools, it is necessary to know HTTP and XML.

The popularity of REST nowadays can be seen in a list of users whose websites and services are based on it. The most popular being Facebook, Twitter, LinkedIn Yahoo, Flickr, some Amazon and e-Bay services, Atom, Google

Glass API and even Tesla model S for communication between the cars and the application.

SOAP is being used by big companies for integration of big number of applications and services, and for integration with older systems, etc.

Considering SOAP users, the biggest one is Google which uses SOAP for implementation in most of its applications, PayPal, Amazon, eBay and other large users such as banking and financial institutions.

This chapter contains a short overview of the most important characteristics and requests of REST architecture and SOAP protocol that become noticeable during use and it is necessary to know them before their practical application. They represent a simple overview according to which we can stress out the most important advantages of each of those technologies.

III. REST

RESTful web services are stateless, meaning the server does not keep session data or any kind of data about previous interactions.

For most servers RESTful web services provide a good infrastructure for temporary storage, which might improve the performances if the returned information does not change very often and it is not dynamic.

Designers and users must understand the context and the contents that is being exchanged because there is no standardized set of rules for describing the REST service interface.

REST is useful for devices with a limited user access (mobile devices) for which the headers are smaller.

REST services can integrate themselves in a more simple way into the existing internet websites, and they are exposed to XML so that HTML sites can consume them more easily. It is not necessary to do everything from the beginning - we just have to upgrade the existing functionality.

REST implementation is much simpler and faster than the one at SOAP

It uses less data and resources during data exchange It works only on HTTP protocol.

A readable JSON data format can be used, as well as normal text, HTML, etc.

It is much faster to learn how to use it compared to SOAP

RESTful web services inherit safety measures from the transport protocol.

IV. SOAP

WSDL (Web Services Description Language) describes a common set of rules for defining messages, links, operations and service locations. It is similar to the contract that defines the interface offered by the service.

SOAP demands a smaller amount of additional code than REST design (such as transactions, safety, coordination, addressing and trust). Most of applications are not simple and they support complex operations that demand conversational state and keeping of contextual information, which makes SOAP better than REST because it does not require any additional encoding.

SOAP web services such as JAX-WS are useful for asynchronous processing.

SOAP supports a few protocols and technologies, including WSDL, XSD and WS-addressing.

It is much more mature from REST, it supports more development tools.

It is more difficult to use it on certain platforms, such as JavaScript.

It has inbuilt error controls.

SOAP defines its safety measures.

Regardless the fact that SOAP protocol is being used more widely, it is being implemented for a longer time, it supports a larger number of auxiliary tools and it possesses numerous positive characteristics, the designers' habits might change in favor of REST architecture due to its simplicity, speed of learning, faster data transfer and more and more advanced possibilities. In the last few years there has been a sudden increase in using application programming interfaces based on REST architecture which can be seen in the biggest API directory on the Internet, ProgrammableWeb**.**

Table 2: Distribution of API protocols and styles based on directory of APIs listed at Programmable Web [6]

YEAR	REST	SOAP	Other
2006	58%	29%	13%
2016	84%	10%	6%

Table 3: Distribution of API protocols on the web confirm the REST trend

API	REST	SOAP
Amazon S3	X	X
Amazon EC2	X	
Facebook	X	
Google Cloud, Maps, applications, Youtube	X	
Twitter	X	
Paypal	X	X
Instagram	X	
Pinterest	X	
LinkedIn	X	
TripAdvisor	X	
Expedia Affiliate Nwtwork	X	X

V. CONCLUSION

After studying the structure and the design of SOAP protocol and REST architecture, the conclusion is that SOAP is quite more complicated to use and a lot more knowledge and code is necessary in order to implement it into web services. REST can also be quite complicated for implementation on the server side, so that when choosing between the two, it must be taken into consideration which one is easier for the application or service development team.

SOAP protocol has from its beginning introduced big changes and progress in communication between different services and protocols and it has greatly sped up and facilitated the idea about the Internet of things.

The principles of REST architecture are being upgraded on the idea of the Internet of things and they support the open web philosophy. When we talk about usage, it is the best to use SOAP in cases when clients need to access the objects that are on servers and there should be an official contract between the client and the server, while REST is better when both the clients and the servers work in WEB surroundings and the client does not need information on objects.

In real world, REST is mostly used in social network services, social networks themselves, services for network data exchange and mobile services, while SOAP is mostly used for financial services, payment services and telecommunication services.

The best way to choose between REST and SOAP is to learn about their advantages and disadvantages in specific surroundings. The first thing that needs to be studies is the purpose and the way of using an application or a service that is being designed, and only after the full scope of the project has been defined, an informed decision can be made.

VI. REFERENCES

[1] M. Masse; “Rest api design”; O’Riley Media Inc., 2012.
[2] L. Richardson and S. Ruby; “Resful web services”; O’Riley Media Inc., 2007.
[3] D. Tidwell, J.Snell and P. Kulchenko “Programming Web Services with SOAP”; O'Riley Media, Inc., 2010.
[4] J. Webber, S. Parastatidis, I. Robinskon; “Rest in practice: Hypermedia and systems architecture”; O’Riley Media Inc., 2010
[5] E. Wilde and C. Pautasso; “REST: From Research to Practise”; Springer, 2011.
[6] J. Wagner; (2014, Jun). Modern API Architectural Styles Offer Developers Choices; ProgrammableWeb; Available: https://www.programmableweb.com/news/modern-api-architectural-styles-offer-developers-choices/2014/06/13; 29.04.2017.