FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Antti Männikkö

WS-* Web Services and Their Suitability for Modern Web Applications

Bachelor's Thesis
Degree Programme in Computer Science and Engineering
January 2020

Männikkö A. (2019) WS-* Web Services and Their Suitability for Modern Web Applications. University of Oulu, Degree Programme in Computer Science and Engineering. Bachelor's Thesis, 22 p.

ABSTRACT

WS-* services are older generation web services that follow the specified Web Service (WS-) standards. The popularity of REST has made these protocols less used and discussed but there are some cases where they are still useful. This thesis goes through five arguably most common web service protocols - XML-RPC, SOAP, WSDL, UDDI and JSON-RPC and discusses their differences and potential use cases today.

Keywords: WS-*, web service, remote procedure call, SOAP, WSDL, UDDI

Männikkö A. (2019) WS-* web-palvelut ja niiden käyttö nykyaikaisissa sovelluksissa. Oulun yliopisto, tietotekniikan tutkinto-ohjelma. Kandidaatintyö, 22 s.

TIIVISTELMÄ

WS-*-palvelut ovat vanhemman sukupolven web palveluita, jotka noudattavat Web Service (WS-) standardeja. RESTin suosio on tehnyt näiden protokollien käytöstä ja keskustelusta vähempää, mutta on olemassa joitakin tapauksia, joissa ne ovat hyödyllisiä. Tämä tutkinto katsoo viittä web-palvelu protokollaa - XML-RPC, SOAP, WSDL ja JSON-RPC - ja pohtii niiden eroavaisuuksista ja mahdollisista käyttötarkoituksista

Avainsanat: WS-*, web-palvelu, etäproseduurikutsu, SOAP, WSDL, UDDI

TABLE OF CONTENTS

FOREWORD

The rapid advancement of web technologies makes it difficult to keep up with current trends. With the abundance of new web frameworks and tools, it is sometimes good to look in the past and see how things have changed, and in some cases, find a new use for old technology.

Oulu, 03.02.2020

Antti Männikkö

ABBREVIATIONS

HTTP	Hypertext Transfer Protocol
XML	Extensible Markup Language
RPC	Remote Procedure Call
JSON	Javascript Object Notation
REST	Representational State Transfer
WS-	Web Standard-
IoT	Internet of Things
API	Application Programming Interface

1. INTRODUCTION

Web development and design paradigms have come a long way since the emergence of the web in the early 90s. Modern web applications are not just .html and .css-files deployed as frontend together with a basic backend. Tools, such as web frameworks, Object-relational mapping (ORM) and JavaScript have helped developers to create more dynamic and user-friendly web applications. At the same time, expectations on what a web application should look like and what functions it should bear have slowly risen

One of the technologies that does not gather as much discussion as other web technologies are web services. Originally, web services were meant for communication between two computers over the Internet but since then their usage has expanded to other gadgets such as mobile and IoT devices.

The current dominance of REST, an architectural style that utilized web resources, has made older services less obsolete, and why not? REST is relatively simple to understand, is platform independent and is pretty fast performance wise. However, WS-* - a series of web service specifications - services are still used in some environments, and REST is still compared to SOAP.

Web development is a constantly evolving field and new technology comes and goes in a very fast pace. In contrast to other technologies, the web services have evolved very little but still maintained a noticeable relevancy. Web APIs have been the buzz for a while and protocol calls between older (possibly legacy) and never systems can use both REST and XML-style approaches between the two.

Web services are widely used but the definition of what web service varies when asked [12]. World Wide Web Consortium (W3C) defines web service as *"...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*" [13].

1.1. Motivation and Scope

This thesis is a literature review and a future-probing discussion of web service protocols that, to at least some degree, fit in this description: XML-RPC, SOAP, WSDL, UDDI and JSON-RPC.

Most of the mentioned web services are 20 years or more old, representing the first generation of web service protocols. How has the architecture stood out the test of time and what is their legacy? The motivation for this thesis is to find alternatives to REST services and find use cases where they are better than REST - and in the case of there being none, find out why the service has fallen out of favor.

First is the introduction of the web service protocols, with examples of structure. The next chapter makes some comparisons within the services and goes deeper on some areas - usage, security and popularity. After that comes discussion, where the future and possible use cases as well as what are WS-* services strengths and weaknesses are discussed. Lastly, the conclusion chapter summarizes previous chapters.

While REST is mentioned in many places, the main purpose of this thesis is not strict comparison of the two. Rather, REST is mainly mentioned to give the reader insight on how the web services have progressed and the transition of technology over the short history of web technologies. Finally, we conclude the thesis with a speculation of what might be a plausible future for the older technologies that still seem to have a place in the world.

2. WEB SERVICE PROTOCOLS

This chapter describes the services and presents examples of services. Many, if not the most, older systems remote procedure calls (RPC). In a nutshell, the client sends a message containing details of what it wants, and the server handles the code implementation part and sends it back via message, also known as request-response messaging. As a simplified example of this, the client wants to know what 2 + 2 is and sends it via message, the server does the calculation and sends the answer 4 back to client.

2.1. XML-RPC

XML-RPC is a simple protocol to make RPC over HTTP. As the name implies, the request and return bodies are in XML, a human readable markup language that is widely used in messaging [7]. It is an early iteration of messaging protocols, from which many more robust and advanced web services have evolved.

```
<?xml version="1.0"?>
<methodCall>
   <methodName>myMethod</methodName>
   <params>
      <param>
         <value><string>"Hello World!"</string></value>
      </param>
   </params>
</methodCall>
```

Figure 1. An example of request call in XML-RPC [7].

2.2. Simple Object Access Protocol (SOAP)

SOAP - created by Microsoft during the late 90s, now maintained byW3C [3] - is a general messaging protocol. It is a deviation of the XML-RPC, as it uses the same protocol for messaging and RPC. Transporting works through HTML-protocol [1].

SOAP consists of three different elements, one of which is optional. Every message has an Envelope element that informs that the XML-document is a SOAP-message and every other element is a child to this element. Body element contains the message itself and may contain any amount of child elements, including the Fault element that has a relevant error message. The optional Header element has contextual information that is needed for processing purposes, and it is always the first child of Envelope [2].

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  ...
 </env:Header>
 <env:Body>
  ...
 </env:Body>
</env:Envelope>

<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
 </env:Header>
 <env:Body>
  <m:myMethod xmlns:m="http://example.org/alert">
   <m:msg>Hello World!</m:msg>
   <m:key>aa123</m:key>
  </m:myMethod>
 </env:Body>
</env:Envelope>
```

Figure 2. The basic structure of SOAP-message and an example message with myMethod-method [2].

The biggest use of SOAP is for RPC purposes. There are few requirements before use: a way to transport the parameters between the SOAP and application and the parameters names and order must always correspond [1, 2]. For example, in order to use the example message Figure 2, we would have to call a method myMethod(msg, key) with myMethod("Hello World!", "aa123").

2.3. Web Service Description Language (WSDL)

WSDL was created by Microsoft and IBM for the purpose of endpoint communication - in other words, it tells what is needed in order to communicate with network service. Like SOAP, it is XML-based and is also maintained by W3C [1, 4].

A common WSDL template has four informational elements. Types have the data type information used by the service, Interface (known as PortType in version 1.1) lists possible operations, Binding describes the transfer protocol and Service has the information on endpoints [4].

```
<description>
  <types>
  </types>

  <interface>
  </interface>

  <binding>
  </binding>

  <service>
  </service>

</description>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:description
    targetNamespace="http://example.org/TicketAgent.wsdl20"
    xmlns:xsTicketAgent="http://example.org/TicketAgent.xsd"
    xmlns:wsdl="http://www.w3.org/ns/wsdl"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/ns/wsdl http://www.w3.org/2007/06/wsdl/wsdl20.xsd">

    <wsdl:types>
        <xs:import schemaLocation="TicketAgent.xsd"
                   namespace="http://example.org/TicketAgent.xsd" />
    </wsdl:types>

    <wsdl:interface name="TicketAgent">
        <wsdl:operation name="listFlights"
                        pattern="http://www.w3.org/ns/wsdl/in-out">
            <wsdl:input element="xsTicketAgent:listFlightsRequest"/>
            <wsdl:output element="xsTicketAgent:listFlightsResponse"/>
        </wsdl:operation>

        <wsdl:operation name="reserveFlight"
                        pattern="http://www.w3.org/ns/wsdl/in-out">
            <wsdl:input element="xsTicketAgent:reserveFlightRequest"/>
            <wsdl:output element="xsTicketAgent:reserveFlightResponse"/>
        </wsdl:operation>
    </wsdl:interface>
</wsdl:description>
```

Figure 3. A basic structure of WSDL template and an example .wsdl-file (source: WSDL 2.0 documentation) [4].

Figure 3 shows an example of TicketAgent.wsdl [4]. Types tell that there is XML-scheme, whereas the interface lists two operations, listFlights and reserveFlights, with their own input and output.

2.4. Universal Description, Discovery, and Integration (UDDI)

UDDI is a business-focused web service that is based on SOAP and WSDL, maintained by Organization for the Advancement of Structured Information Standards (OASIS). Its idea is to find a client service either by querying or browsing [5]. Sometimes referred to as "phone directory" of web services, UDDI has three key components called pages. White pages contain information such as names, addresses and contacts. Yellow pages have the information on service in a categorized form. Lastly, green pages have the technical information of service [1].

```
<businessEntity businessKey = "..."
   operator = "http://www.myfictionalservice.com" authorizedName = "John Doe">
   <name>My Fictional Service </name>
   <description>
      Description of my service
   </description>
   <contacts>
      <contact useType = "general info">
         <description>General Information</description>
         <personName>John Doe</personName>
         <phone>...</phone>
         <email>...</email>
      </contact>
   </contacts>
   <businessServices>
    ...
   </businessServices>
   <identifierBag>
    ...
   </identifierBag>
   <categoryBag>
    ...
   </categoryBag>
</businessEntity>
```

Figure 4. An example of the UDDI businessEntity element [1].

The businessEntity element (as seen in figure 4) is the main entity on UDDI. In addition to containing information such as white page, it has a businessService element to display provided service, categoryBag for further categorization of services and identifierBag for a non-contact identifier such as D-U-N-S number [5].

```
<tModel tModelKey = "..." operator = "http://www.myfictionalservice.com"
   authorizedName = "John Doe">
   <name>MyFictionalService Interface</name>
   <description>
      Interface definition for MyFictionalService
   </description>

   <overviewDoc>
      <overviewURL>
         http://www.myfictionalservice.com/myinterface.wsdl
      </overviewURL>
   </overviewDoc>
</tModel>
```

Figure 5. An example of a tModel template [1].

Other mentionable entities of UDDI are bindingTemplate and tModel. bindingTemplate has technical information for applications, such as service key. tModel, on the other hand, is used for describing concepts in the UDDI registry. Example of tModel usages are protocol definitions or categorization groups [5].

2.5. JSON-RPC

During the 21st Century, Javascript has risen as a de facto language to web development. In conjunction to that, the interest to JSON as a data format has also

become more popular. JSON-RPC was designed as a simple alternative to XML-RPC and it shares many similarities, such as error codes [6].

```
--> {"jsonrpc": "2.0", "method": "myMethod", "params": ["Hello, "World!"], "id": 1}
<-- {"jsonrpc": "2.0", "result": "Hello World!", "id": 1}
```

Figure 6. An example of JSON_RPC objects, the first being Request and second being Response [6].

JSON-RPC Request object has four parts: jsonrpc, which specifies the used version; method, which contains the name of a wanted method; params to the method and id which can be a string, number or null. The Response object has response or error member, depending on the success on a method call, on top of jsonrpc and id. Unlike in Request case, id is not optional in Response and must match with REquest objects id [6].

3. COMPARISON

This chapter compares differences in web services described in previous chapter.

3.1. Usage

The intended usage of web service is not always clear. Understanding what a service can and cannot do saves a lot of headache from the developer.

Table 1. How different web services are used.

Usage				
XML-RPC	SOAP	WSDL	UDDI	JSON-RPC
Simple messaging	Communication	Description	Discovery	Simple messaging

XML- and JSON-RPC's advantages are their simplistic nature. The downside of this is that they are less optimization for different environments [20]. XML-RPC and JSON-RPC are rarely used nowadays; an example of where both could shine is learning environment where students are less experienced with messaging with RPCs.

One of the reasons for SOAP's longevity in web development is its platform independence. If client implementation can handle XML, it can communicate with a SOAP server. Almost every programming languages have an implementation of SOAP framework, including Python, Java or C#.

```
from zeep import Client

client = Client("http:/www.myfictionalservice.com/myinterface.wsdl")
result = client.service.myMethod("Hello", "World")
```

Figure 7. Using Zeep Python library to read WSDL file and message to the SOAP server [30].

WSDL main function is to provide information on web service. WSDL service has two important pieces of information: the abstract interface of the application and specific information on end-point connection [1].

UDDI's idea is to connect businesses all over to world together. Using UDDI's registry, the user could find potential service he or she needs. UDDI has been criticized for various issues, such as scalability and bottlenecking [21].

3.2. Security and Error Handling

Over the years, web services have been susceptible to different types of security attacks. XML without encryption makes it relatively easy to access and decipher information [22]. Another well-known XML exploit is to send an oversized file, creating an XML Denial of Service (XDoS) attack [22]. SOAP's routing can be changed to allow Denial of Service (DoS) and man-in-the-middle attacks [23].

WSDL is vulnerable to parameter tampering, where users can try to get sensitive information by changing the parameters, as well as user scanning WSDL document and finding sensitive information such as port types [24].

Despite the potential issues, SOAP has been as a secure protocol. WS-Security protocol was published early on its lifecycle, and it contains methods such as message integrity, confidentiality and authentication [25].

Error handling and fault tolerance are something that everyone who works in programming field appreciates - completely bug-free programs are sparse and in the event of bug or crash, it is nice to know what exactly triggered it. While web services cannot fix the underlying problem - that job is usually left to frameworks - it can send a response element containing the error Table 2.

Table 2. Fault Handling elements on various services.

Fault Handling				
XML-RPC	SOAP	WSDL	UDDI	JSON-RPC
<fault> element	<fault> element	SOAP <fault> element	SOAP <fault> element + error codes	Error member in Response

Error handling elements are pretty much the same in XML-based services. The Fault element consists of error code with an error message. This makes debugging slightly easier.

There are debates if SOAP's fault tolerance is good enough. Many have theorized and developed a different style of fault managers [8, 9] or middleware element [10,11] for SOAP to give more robust fault tolerance.

3.3. Popularity

While generally on the downswing, services like SOAP are still used in applications like distributed computing and banking services due to its security, legacy support and asynchronous requesting [19].

Table 3. Amount of Google searches per service in million (source: Google Trends).

Google Searches					
Date (M-Y	XML-RPC	SOAP	WSDL	UDDI	JSON-RPC
01-2004	45	66	67	13	0
01-2009	13	25	43	16	18
01-2014	1	14	24	0	5
01-2019	3	8	10	<1	8

As Table 3 shows, the trending towards WS-* web services has been on decline as the 21st century has moved on. Possible reasons for this could be demanding requirements to run them [14], not being loosely coupled as REST [15], being complex and web services expanding to more areas than just business applications [16].

UDDI has never been as popular SOAP or WSDL. Part of its massive drop in searches in 10's is that major organizations such as Microsoft terminating UDDI

services on their servers [17]. However, implementations of the service such as jUDDI are still developed and maintained [18].

In case of XML- vs JSON-RPC, the former has fallen in terms of popularity, while the latter has kept up some amount, although still small, of relevance. Javascript's relevance in web development has raised the usage of JSON-format which explains more interested in JSON-RPC.

Table 4. Amount of Google Searches of REST and RESTful (source: Google Trends).

REST Google Searches		
Date (M-Y)	REST	RESTful
01-2004	15	8
01-2009	36	31
01-2014	59	69
01-2019	90	73

Whereas the trending of WS-* services have gone down, REST and RESTful services have become more popular over time. In comparison, SOAP had 8,8% and WSDL 11,1% amount of Google searches compared to REST in January 2019.

4. DISCUSSION

The WS-* specified web service protocols have not fared well against more recent solutions such as REST. For a simple web page, these kinds of services are either far too complex or too demanding on hardware. However, they bear some use cases that might interest web developers and designers.

Bandwidth is one the restrictive factors for web development. WS-* services are particularly taxing on bandwidth with its metadata communication and large XML-files. In case of very limited bandwidth, services like SOAP is not practical to implement.

HTTP has been the de facto transport protocol for a while now. However, SMTP and less popular protocols like Java Message Service (JMS) are still used. RESTful APIs mostly communicate with HTTP methods, and while it is possible to do a something like an email implementation, the developer should consider WS-* for their more inclusive selection of transfer protocol implementations.

WS-* services have some security issues - but in the end, the standardization has helped to prevent critical cases happening. Very sensitive information such as banking credentials would be better in the hands of WS-Security compared to what REST offers [26]. Financial organizations like banks still prefer SOAP and WSDL over REST because of the formal contract between client and server.

XML- and JSON-RPC are not developed enough to attract developers but could be used in learning purposes. Their basic architecture could visually show how messaging and RPC works.

The future for UDDI does not look bright. Bigger organizations terminating their services and overly low interest in the recent years points that the service is not worth exploring. However, in case of personal or maybe even organizational interest, modern implementations are still found.

Mobile development has steadily become one of the biggest part of IT-industry in terms of users and revenue. While possible, WS-* services are very rarely used in mobile services due their heaviness and bandwidth consumption.

The biggest question of WS-* services relevancy right now is the relevancy of XML as a payload format. JSON has become more popular over time, and more contenders like YAML have also appeared. What does a verbose format like XML could offer that its counterpart lack? In the future, a web service that offers the same qualities as SOAP but uses JSON could easily mean that SOAP is no longer more than a relic from the past of web development.

Another big question is future support of WS-* APIs and frameworks. For example, Amazon S3 recommends using REST API over SOAP, and future features will not support SOAP [27]. Some organizations, such as PayPal still support both SOAP and REST.

The idea of SOAP and WSDL getting more popular is not still completely buried. The last changes to WSDL standardization made it easier to implement [28] but it at a time of this writing it has been 12 years since last revision. Making the standard more compatible with current technologies, as well as making it easier to communicate with REST based systems would probably grow more interested towards the service. W3C is still very active and could further implement these systems forward if there is interest.

One field that will need WS-* service experts are legacy services. In some fields, such as financial or military, developing new system would take years to design and

develop and would be costly, making it better to use older technology for development. Even so, workarounds such as implementing REST type calls that include SOAP message inside the call are plausible if new systems are developed to support the legacy system.

The good news for a developer that wants WS-* protocol in their architecture is that there are frameworks that implement SOAP and WSDL and are still in development. For example, Microsoft's .Net framework offers many WS-* specifications and supports SOAP and WSDL [29].

5. CONCLUSION

The popularity of WS-* services has dropped since REST architecture came popular. This does not mean said services are dead - services like SOAP and WSDL have their uses.

XML- and JSON-RPC are too simplistic for true use, and therefore not relevant in the current context of web development. Likewise, UDDI services have become less and less popular as years have moved despite development still going. Major organizations dropping the support for the service points that there is not much demand for the service.

SOAP, like other services, has also suffered from newer services appearing in the market, but it still has some relevance. Platform independence and implementations for many languages still attract some designers and developers, particularly those who do not work with the HTTP protocol.

Legacy systems still hang on WS-* services, and while it is possible to implement interfaces that messages between the old and new technology, experts on the older web service technology field are still needed.

Object-oriented languages like Java, C# and C++ are widely used. For these languages, the description of what services contain is very important in order to avoid errors. WSDL helps developers and designers to understand the services better and help on implementation.

The future for WS-* style services is dependent on current technology. If for some reason, XML as a format would disappear in favor of JSON or some other format, it would mean bad time for older services that rely on XML. In the case of this happening, the services would need to be redefined or a new service that replaces them need to be implemented.

6. REFERENCES

[1] Curbera, Francisco, et al. "Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI." *IEEE Internet computing* 6.2 (2002): 86-93.

[2] Snell, J., Tidwell, D., & Kulchenko, P. (2001). *Programming web services with SOAP: building distributed applications*. " O'Reilly Media, Inc.".

[3] Hadley, M., Mendelsohn, N., Moreau, J., Nielsen, H., & Gudgin, M. (2003). SOAP Version 1.2 Part 1: Messaging Framework. *W3C REC REC-soap12-part1-20030624, June*, 240-8491.

[4] Chinnici, R., Moreau, J. J., Ryman, A., & Weerawarana, S. (2007). Web services description language (wsdl) version 2.0 part 1: Core language. *W3C recommendation*, *26*(1), 19.

[5] Bellwood, T., Clément, L., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y. L., ... & von Riegen, C. (2002). UDDI Version 3.0. *Published specification, Oasis*, *5*, 16-18.

[6] JSON-RPC Working Group. (2012). Json-rpc 2.0 specification.

[7] Kidd, E. (2001). XML− RPC HOWTO. *Tech. Rep.*

[8] Liang, D., Fang, C. L., Chen, C., & Lin, F. (2003, December). Fault tolerant web service. In *Tenth Asia-Pacific Software Engineering Conference, 2003.* (pp. 310-319). IEEE.

[9] Dialani, V., Miles, S., Moreau, L., De Roure, D., & Luck, M. (2002, August). Transparent fault tolerance for web services based architectures. In *European Conference on Parallel Processing* (pp. 889-898). Springer, Berlin, Heidelberg.

[10] Santos, G. T., Lung, L. C., & Montez, C. (2005, September). Ftweb: A fault tolerant infrastructure for web services. In *Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05)* (pp. 95-105). IEEE.

[11] Merideth, M. G., Iyengar, A., Mikalsen, T., Tai, S., Rouvellou, I., & Narasimhan, P. (2005, October). Thema: Byzantine-fault-tolerant middleware for web-service applications. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)* (pp. 131-140). IEEE.

[12] Hanna, S., & Munro, M. (2008, April). Fault-based web services testing. In *Fifth International Conference on Information Technology: New Generations (itng 2008)* (pp. 471-476). IEEE.

[13] World Wide Web Consortium. Web Services Glossary. Retrieved 1 March 2019 from https://www.w3.org/TR/ws-gloss/

[14] Guinard, D., Trifa, V., Pham, T., & Liechti, O. (2009, June). Towards physical mashups in the web of things. In *Proceedings of INSS* (Vol. 9, pp. 17-19).

[15] Pautasso, C., & Wilde, E. (2009, April). Why is the web loosely coupled?: a multi-faceted metric for service design. In *Proceedings of the 18th international conference on World wide web* (pp. 911-920). ACM.

[16] Guinard, D., Ion, I., & Mayer, S. (2011, December). In search of an internet of things service architecture: REST or WS-*? A developers' perspective. In *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services* (pp. 326-337). Springer, Berlin, Heidelberg.

[17] Microsoft. What's New in BizTalk Server 2013 and 2013 R2. Retrieved 5 March 2019 from https://docs.microsoft.com/en-us/biztalk/install-and-config-guides/what-s-new-in-biztalk-server-2013-and-2013-r2

[18] jUDDI. Retrieved 5 March 2019 from https://juddi.apache.org/index.html

[19] Tihomirovs, J., & Grabis, J. (2016). Comparison of soap and rest based web services using software evaluation metrics. *Information Technology and Management Science*, *19*(1), 92-97.

[20] Allman, M. (2003). An evaluation of XML-RPC. *ACM sigmetrics performance evaluation review*, *30*(4), 2-11.

[21] Al-Masri, E., & Mahmoud, Q. H. (2008, April). Investigating web services on the world wide web. In *Proceedings of the 17th international conference on World Wide Web* (pp. 795-804). ACM.

[22] Moradian, E., & Håkansson, A. (2006). Possible attacks on XML web services. *IJCSNS International Journal of Computer Science and Network Security*, *6*(1B), 154-170.

[23] Demchenko, Y. (2004). "White collar" Attacks on Web Services and Grids. *Grid Security threats analysis and Grid Security Incident data model definition, Draft Version 0.2*.

[24] Lindstrom, P. (2004). Attacking and defending web services. *a spire research report*.

[25] Rosenberg, J. B., & Remy, D. L. (2004). *Securing web services with WS-security: Demystifying WS-security, WS-policy, SAML, XML signature, and XML encryption*. Sams.

[26] Wagh, K., & Thool, R. (2012). A comparative study of soap vs rest web services provisioning techniques for mobile host. *Journal of Information Engineering and Applications*, *2*(5), 12-16.

[27] Amazon. Appendix: SOAP API. Retrieved 18 March 2019 from https://docs.aws.amazon.com/AmazonS3/latest/API/APISoap.html

[28] Chinnici, R., Haas, H., Lewis, A. A., Moreau, J. J., Orchard, D., & Weerawarana, S. (2007). Web services description language (WSDL) version 2.0 part 2: Adjuncts. *W3C Recommendation*, *6*.

[29] Microsoft (2017). Web Services Protocols Interoperability Guide. Retrieved 19 June 2019 from https://docs.microsoft.com/en-us/dotnet/framework/wcf/feature-details/web-services-protocols-interoperability-guide

[30] Zeep: Python SOAP client. Retrieved 26 January 2020 from https://python-zeep.readthedocs.io/en/master/