A Comparative Study of REST with SOAP

Usman Riaz(✉), Samir Hussain, and Hemil Patel

University of Leicester, University Road, Leicester LE1 7RH, UK
{UR19,SMH65}@Student.le.ac.uk, HP339@Studen.le.ac.uk

Abstract. Currently most web services obtain REST architectural styles, which were originally founded by Roy Fielding in the year 2000. Alongside the styles, constraints and techniques were also discussed in Roys famous dissertation. In this research paper we will take a look at understanding the techniques used in the REST architecture style, covering the six constraints. Evaluating the testing techniques of REST and finally comparing REST with the SOAP standard. In order to remove the high latency, reduce network traffic and processing delays, which was being caused by SOAP, REST was introduced to overcome all of these issues. Furthermore, 70% of websites use REST architecture as many have found SOAP to be outdated.

Keywords: REST · SOAP · Web-services

1 Introduction

Often REST is considered into two parts: web services and API.

Web services: a service that is offered over the web through standardized web protocols, via the use of data exchange systems, often formatted through XML. The key feature of web services is that your able to write them in various languages, but you are still able to communicate by exchanging data (Cauldwell et al. n.d.).

Application programming interfaces are self-explanatory whereby an interface, programmed with software, can interact with an existing application. Expanding further, an API allows you to build upon information and functionality via a set of functions and procedures. An example of everyday API use would be when logging into a website via your Facebook profile (Kopecký et al. 2008).

There is a key difference between both, web services require a network, compared to API's which can be both on and offline. Web services are not open source meaning the understanding comes through JSON or XML. On the other hand, API's are open source. An interesting aspect of web services is that they are not lightweight and hence often require SOAP to send and receive messages whereas API's are a lightweight architecture (Roy Thomas Fielding 2000).

W. Fu et al. (Eds.): ICMTEL 2021, LNICST 387, pp. 485–491, 2021.
https://doi.org/10.1007/978-3-030-82562-1_47

1.1 Research Gap

In this paper we have studied several research papers spanning from the years 2005 till 2020, this is so that the research gap between these years where rest and soap have evolved is discussed thoroughly and the necessary gaps in research have been filled. REST style is now part of many different social media platforms and this part was intriguing to find out as to how REST has taken over our IoT. Most studies were showing generic research of REST and SOAP, whereas this paper delves into how REST and SOAP relate to everyday circumstances.

2 Representational State Transfer

2.1 What Is Rest?

Representational state transfer (REST), a type of software architecture was developed to ensure the exchange of information between different computer systems. The method builds on existing systems which is one of the reasons why it is popular and is also simplistic in its use. Strict constraints are put in place for the development of web services, these services adapt to the REST architecture which allows for better communication with one another (Battle and Benson 2008).

2.2 The 6 Constraints

The REST architectural style is designed upon these six constraints or design rules. These constraints only define the way data is exchanged between components.

Stateless and Client-Server
There is no restriction between client-server communication to protocol. In the early days of web development, it was needed that the server-side of the web application needed to remember certain details about the user that is using it (Duffy 2015). In REST, the client passes to the server everything that it needs to perform the action and returns to the client with the action complete, this is the complete process. By doing this, when requests get fulfilled, there is no need for the server to remember any of the steps. Systems using REST interact using standard operations on resources, this means that they are not reliant on the implementation of interfaces (Abassi 2015). This makes REST less complex and simplistic as discussed because it removes all server-side state synchronization logic, and the server never loses track of each client in the application as the client sends the necessary information with each request they make. All in all, the client and server application must evolve separately without being dependent on one another (Fig. 1).

Layered System
RESTful allows for the use of layered system architecture. Layered systems consist of layers with different units of functionality. These layers can be added, removed, or modified. The layers only communicate with the layer above or below; the above layers rely on below layers to perform functions. You can deploy the API on server A, store

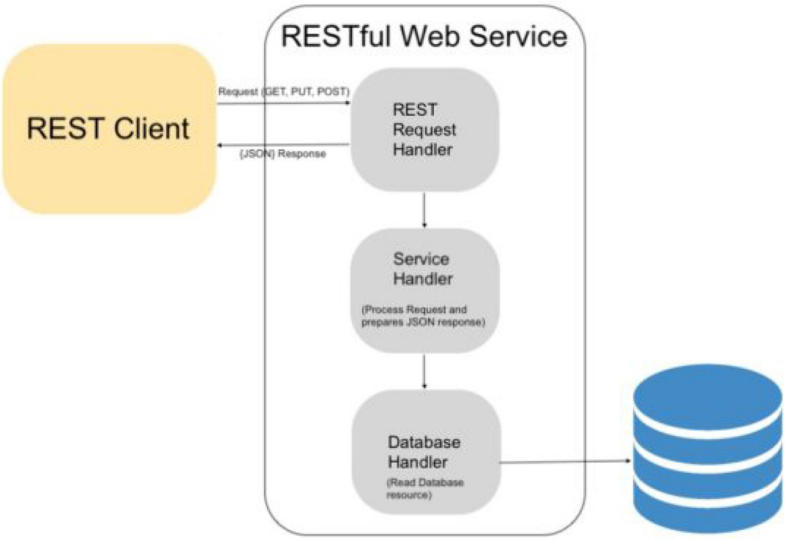

Fig. 1. To show RESTful web Service Architecture (Phppot 2019)

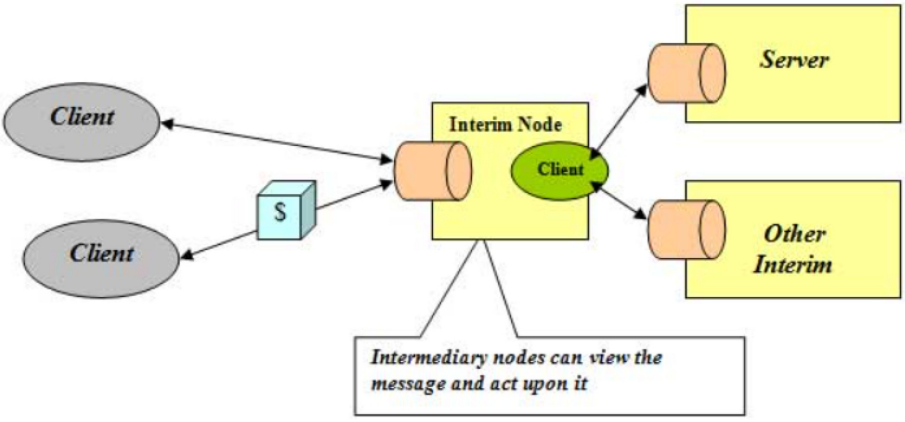

Fig. 2. To show a representation of a layered system (mrbool.com, n.d.).

data on Server B and authenticate requests in Server C. The client cannot talk to the database server directly because it is not the adjacent layer (Abassi 2015) (Fig. 2).

Uniform Interface

The Uniform Interface constraint separates REST from other architecture styles. The uniform constraint decouples the interface, this means that each component can exist and also perform various tasks independently of one another, e.g., being able to read

an article and send emails on the same browser without the need any sort of extension (restfulapi.net, n.d.).

Cacheable
Caching of data will bring performance improvement for the client-side and the load on the server is reduced due to better scope for scalability. When a client requests a resource representation, this request will then go through a cache toward the service hosting that resource. This can either be applied on the server or the client-side, the resources that caching has been applied to need to declare themselves as cacheable (restfulapi.net, n.d.).

Code on Demand
This is an optional constrain in REST. The constraint allows for better flexibility on the client-side and the server decide how certain tasks will be done. The use of code of demand will reduce visibility hence why this constraint is optional. Some examples that influence code on demand are flash and java applets (restfulapi.net, n.d).

3 Testing and Verification Techniques of REST

3.1 HTTP Methods

Several requests are sent to a REST API and verifying responses from it. The purpose of this testing is to record the responses, this is done by sending HTTP requests to see if the REST is working as it should. This is done by using the HTTP request methods, GET, POST, PUT and delete. The GET method is to get information from a server using a URI. This request has no other effect as it is only used for the extraction of data. POST, this request can be used when trying to send data to the server and when trying to create an entity. An example of this action is uploading a file, using HTML forms etc. the PUT method is used to update or replace an existing entity on the server. Lastly, the delete method removes the specified resource given by a URI (W3 schools.com 2019).

A POST writes data to the API, the data can be put into the body of the API request, normally web browser does not allow for data to be put into the body of a request however extensions such as "Postman – REST Client" can be used. An example of using POSTMAN API is being able to send out a tweet over the Twitter API. To send out tweets a form of authentication is required, many big companies use OWASP for authentication. POSTMAN can also be used to run SOAP requests, to do this we first need to get a WSDL sample, it is a web service description language which is used in SOAP. Once WSDL sample has been gained we can take the request URL and run the request using POSTMAN however unlike REST, SOAP is restricted to XML.

3.2 Testing and Verifying

Testing is critical before deployment as during development API testing can reveal issues with your API, server, network and more. The first concern is functional testing, this is to ensure that the API functionals correctly, the main objective is to have no

bugs and make sure that implementation is working as expected and as specified in the requirements (Mor 2019). Functional test cases have the same test actions, whether the testing is automated or manual. Some individual actions a test needs to take include, verifying responsive headers, verify correct HTTP status code and basic performance sanity. Apart from testing headers and bodies, every URL should point to an existing location, a broken URL can cause the hypermedia mechanism to fail (Fertig and Braun 2015).

Security testing, the purpose of this is to discover the vulnerabilities of an application and is the first step of the audit process. Security steps include authentication, authorisation, and encryption. There are principles to security testing in RESTful APIs, these can be implemented into a web server. Some tests include incorrect input type must be rejected, an incorrect input size must be rejected, and API must provide the expected output for a given input value (Atlidakis 2018). The next step of security testing is to do some penetrating testing, this is a deliberate attack in a controlled environment and helps with vulnerabilities that may have occurred during the development of web services. A list of vulnerabilities applicable to the application can be listed e.g., exposing a directory traversal attack with resources such as an image. Finally, any unauthorized access that is made during testing can be filed and patched to stop any future intrusions (Fertig and Braun 2015). The final aspect of testing is Fuzz Testing, this where an API is pushed to its limits with vast volumes of requests being sent to it. This technique consists of finding implementation bugs using malformed data injection in an automated fashion (Godefroid, Levin and Molnar, n.d.).

Fuzz testing is a type of testing to see how the API can handle random data which acts as input data, similar to the load testing this type of testing is constructed in order to verify the absolute limit of the API, the random data is used to carry out a crash or uncover negative behavior in the API (Fig. 3).

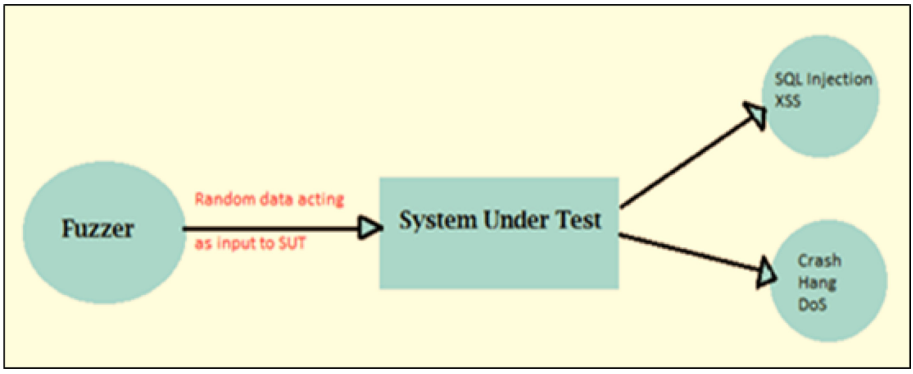

Fig. 3. (Fuzz Testing (Fuzzing) Tutorial: What is, Types, Tools & Example, n.d.)

4 REST Application Compared to SOAP Evaluation

REST is an architecture style, it is more flexible compared to SOAP and does not require processing, on the other hand, the rules in SOAP are important as you cannot achieve any level of standardisation without them. REST seeks to fix the problems with SOAP where soap has been around for some time. SOAP is reliant on XML to provide messaging services and took the place of technologies that were much older and did not work well within the internet (Zur Muehlen et al. 2005). Many developers later discovered that SOAP was complex and hard to use e.g., you need to create the XML structure every time using code to perform simple tasks. Comparing this to REST, web services are reliant on using URL approach and responses do not have to be provided in XML but can use JSON, CSV and RSS. It is easier to parse some forms than others depending on the language being used for the application, this is where REST excels (Soni and Ranga 2019).

A REST API works pretty much normally as a website does, you make a call from a client to a server and you get data back over the HTTP protocol. The best example can be found on Facebook graph API. Searching www.facebook.com/youtube in the google search bar brings up the YouTube page on Facebook. However, if we change the "www" to "graph", the returned result is a response to our API request, a page with JSON formatted data is returned. This is consuming data to API however we can also write data to the API using HTTP request methods. Another example is an application using google translate, this will access google server. It does this through its RESTful API. Words are sent to it according to the language that you have chosen, these words are then returned translated. The list goes on, many services have RESTful interface, including eBay, Flicker, Wikipedia and many more.

In conclusion, SOAP is not able to use REST as it is a protocol compared to REST which is an architectural pattern, however, REST can use SOAP as it is a protocol for web service use. A debate of when to use REST and when to use SOAP when designing a web service is very common. When limited resources are at hand and limited bandwidth, going for SOAP is not a good idea as discussed before the messages include a vast amount of content. This in turn will consume a great amount of bandwidth and therefore REST should be used instead as the messages will mostly consist of JSON messages (Wagh and Thool 2012).

References

Battle, R., Benson, E.: Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST). J. Web Semant. **6**(1), pp. 61–69 (2008)

Duffy, S.: What Is REST? | Scott Duffy (2015). https://www.youtube.com/watch?v=LHJk_ISxHHc . Accessed 22 Oct 2020

Roy Thomas Fielding: UNIVERSITY OF CALIFORNIA, IRVINE Architectural Styles and the Design of Network-based Software Architectures (2000). https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

Abassi, E.: Differentiating Parameters for Selecting Simple Object Access Protocol (SOAP) vs. Representational State Transfer (REST) Based Architecture. [online] researchgate.net (2015). https://www.researchgate.net/publication/280736421. Accessed 22 Oct 2020

Kopecký, J., Gomadam, K., Vitvar, T.: hRESTS: An HTML Microformat for Describing RESTful Web Services. In: 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, Sydney, NSW, 2008, pp. 619–625. https://doi.org/10.1109/WIIAT.2008.379

Mor, R.: REST API Testing Strategy: What Exactly Should You Test? l Sisense. [online] Sisense (2019). https://www.sisense.com/en-gb/blog/rest-api-testing-strategy-what-exactly-should-you-test/. Accessed 26 Oct 2020

Fertig, T., Braun, P.: Model-driven Testing of RESTful APIs. https://doi.org/10.1145/2740908.2743045. http://www.www2015.it/documents/proceedings/companion/p1497.pdf. Accessed 26 Oct 2020

Atlidakis, V., Godefroid, P., Polishchuk, M.: REST-ler: Automatic Intelligent REST API Fuzzing (2018). https://arxiv.org/pdf/1806.09739.pdf. Accessed 26 Oct 2020

Phppot. (n.d.). PHP RESTful Web Service API – Part 1 – Introduction with Step-by-step Example. https://phppot.com/php/php-restful-web-service/. Accessed 26 Oct 2020

Godefroid, P., Levin, M., Molnar, D. (n.d.): Automated Whitebox Fuzz Testing. http://pxzhang.cn/paper/concolic_testing/FuzzTesting.pdf. Accessed 27 Oct 2020

Zur Muehlen, M.V., Nickerson, J., D. Swenson, K.: Developing web services choreography standards—the case of REST vs. SOAP (2005). www.sciencedirect.com. https://d1wqtxts1xzle7.cloudfront.net/. Accessed 28 Oct 2020

Soni, A., Ranga, V.: API Features Individualizing of Web Services: REST and SOAP (2019). www.researchgate.net. https://www.researchgate.net/profile/Virender_Ranga2/publication/335419384. Accessed 28 Oct 2020

Wagh, K., Thool, R.: A Comparative Study of SOAP Vs REST Web Services Provisioning Techniques for Mobile Host (2012). https://www.researchgate.net/. https://www.researchgate.net/profile/Dr_K_Wagh/publication/264227921. Accessed 10 2020

restfulapi.net. (n.d.). REST Architectural Constraints - REST API Tutorial. https://restfulapi.net/rest-architectural-constraints/#cacheable. Accessed 30 Oct 2020

W3schools.com. HTTP Methods GET vs POST (2019). https://www.w3schools.com/tags/ref_httpmethods.asp

mrbool.com. (n.d.). REST Architectural Elements and Constraints. http://mrbool.com/rest-architectural-elements-and-constraints/29339#:~:text=Data%20Elements. Accessed 31 Oct 2020