Modifiability in SOAP vs REST API

Ilian Ayoub, Christian Habib

Linköping University

Introduction

The poster aims to convey the conclusions we came to when investigating how SOAP and REST script-based clients handle change in API. The main point of this study is to see what issues occur on the client side when API is modified.

How well does a script-based client handle a change in a REST service compared to a SOAP service?

Background

SOAP and REST comprise most of the communication architectures present on the web. REST being a simple web communication API over HTTP and SOAP being a communication protocol not constrained to HTTP. SOAP messages use XML as format whilst REST is not constrained.(1)

A SOAP service uses a WSDL to allow clients to interface with it. WSDL is a document that describes the SOAP web service and all its components. All that the client needs to know about the service is described in the WSDL.(1)

REST is relies on URI:s which point to a specific resource where a client can submit or retrieve resources.(2) REST uses a concept called HATEOAS which implies that one can place hyperlinks in a response that enables clients to explore other resources.(3)

Method

We implemented two web clients,one a SOAP client and one a REST client, where both interface with respective SOAP server and REST server. Both clients where implemented in python while the SOAP client was using the zeep library(4) and the REST client used the requests library(5). Both the REST and the SOAP server were implemented using the python web framework Flask.(6)

This study compares the two API implementations in terms of modifiability. In other words, what issues will the client have when the service API is changed and how hard is it for the client to fix these issues. The experiment will consist of changing the API and see what ramifications it will have for the clients and what it takes to fix the issues.

```
r = requests.get('http://0.0.0.0:8080/1')
print(r.text[:200])
```

Figure 1: GET request in the REST client.

```
wsdl = 'http://localhost:5000/soap?wsdl'
client = zeep.Client(wsdl=wsdl)
u = client.service.get_user(1)
print(u)
```

Figure 2: get_user function in the SOAP client.

As shown in the figures above the client implementation differ a bit. The REST client requests the resource by providing the URI of resource and the SOAP client remotely invokes the get_user() method and retrieves the response.

Results

When we changed the location of a resource in the REST server the client received a 404 not found. The client had no way of remedy this because it has no information on where the resource is anymore. When we renamed the SOAP function the client could not invoke that function however the client could at least see in the WSDL file what functions are available.

When changing a POST method in the REST API by removing a parameter the client using the old API will receive a 400 bad request. When removing a parameter in SOAP we get a error message that more resembles a local error which gives us information about the fault. From the WSDL document we can see what parameters a given function takes and fix the issue.

Discussion

Our tests focused on a basic implementation of a REST and SOAP API and how a script-based client using these services would be affected by the API changing. We found that in this scenario a SOAP API would benefit the client the most.

REST relies heavily on HATEOAS to navigate the user through resources, however when the client is not a browser where users can simply click on a hyperlinks, browsing resources becomes difficult. SOAP gives you all the functions and everything you need to know about the service to be able to interface with it. This makes it much easier for a client to be aware of any changes made to the API.

For a REST API to be useful to a client extensive documentation about the resources and where to find them is needed for the API to have value to the client.

We find that from a script-based client's point of view the easier code to maintain would be the one interfacing with a SOAP service.

References

(1) https://www.soapui.org/learn/api/soap-vs-rest-api.html
(2) Fielding, Roy T., and Richard N. Taylor. *Architectural styles and the design of network-based software architectures*. Vol. 7. Doctoral dissertation: University of California, Irvine, 2000
(3) https://restfulapi.net/hateoas/
(4) https://python-zeep.readthedocs.io/en/master/
(5) https://realpython.com/python-requests/
(6) Vogel, Patrick, et al. "A Low-Effort Analytics Platform for Visualizing Evolving Flask-Based Python Web Services." 2017 IEEE Working Conference on Software Visualization (VISSOFT). IEEE, 2017

LINKÖPING UNIVERSITY