Web Services and Web APIs

Martin Zuber, B00066378

Module COMP H4029

March 21, 2016

1 Abstract

This paper is a literature review of various online articles discussing the web services (WS) and web application programming interfaces (Web APIs). The aim was to study web services build on top of Simple Object Access Protocol (SOAP) and currently very popular Representational State Transfer (REST) like Web APIs. I will start with a short description and comparison of both technologies. Basically, what they are and how they differ without going into the details and without reciting the specifications. I will then continue with the identification of some examples of appropriate use cases for each technology in general. At last, I will explore the evolution of both technologies and their current position in the contemporary ecosystem of the web and I will provide a real life example of a great Web API implementation.

2 SOAP vs REST

Simple Object Transfer Protocol (SOAP) is a protocol specification. Originally developed by Microsoft in 1999.[1] It specifies how to exchange an information between two applications. SOAP uses service interfaces to expose the business logic. Extensible Markup Language (XML) is used for information formatting and any application layer protocols for information transmission. SOAP protocol defines it's own security measures and have built-in error handling feature.

Representational State Transfer (REST) on the other hand is an architectural style definition, a set of constraints for a network based application design. It was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. It uses Uniform Resource Identifiers (URIs) to expose the business logic of the application. The information formatting is not constrained but JavaScript Object Notation (JSON), XML or Rich Site Summary (RSS) is widely used. REST is defined as a protocol independent but uses HTTP exclusively for the information transmission. This is a result of HTTP being the protocol of the Web. REST doesn't define any security measures and therefore, relies on security capabilities of underlying transmission protocol. Because REST is not a protocol, but an architectural style definition, not all its constraints must be implemented. Public facing APIs usually refer to them self as REST-full or REST-like, where REST-full API would follow all the REST constraints and REST-like only some.[2] Unfortunately, REST is being often misinterpreted and not implemented according to its definition. Many of the "REST-full" Web APIs are in reality close to an REST-like services.

Probably a most important difference between SOAP and REST lie in the degree of coupling between the client and server implementations. With SOAP, the client has a rigid contract with the service. The client must have previous knowledge about the functionality of the service it is consuming. This is accomplished with the usage of Web Services Description Language (WSDL). WSDL is designed to be machine-readable and defines the interface which a client will be using to communicate with the service. The client is therefore very specific for the application and even slight change in the service implementation would break this tight coupling.

On the other hand, a REST client is more generic. If a REST service is done right, a client doesn't need to have any previous knowledge about the service it wants to consume. All the client needs is an entry point. With each interaction with the service, the client will learn more about the service interface. This is accomplished by enforcement of Hypermedia as the Engine of Application State (HATEOAS) constraint of REST architecture specification. In each response, the server will include a list of URLs pointing to the functionality related to a request client just made. For example, if a client requests a resource creation, the service will respond with a message containing (among other information) a link where the created resource was created and how to edit or remove the resource and so on. This approach allows for very loose coupling between the client and the service. This way the server functionality evolution is independent of the clients and can be handled without breaking their functionality.

3 When to use SOAP and REST

The SOAP protocol has various advantages over the REST architecture as mentioned earlier in this document. One of the main advantages is independence from the underlying transport protocol. Soap can be implemented on the top of HTTP, FTP, SMTP or even Java Messaging Service (JMS). It's important to note some other strengths of the protocol. Namely SOAP is pretty mature, well-defined protocol which comes with complete specification.[5]

These characteristics make SOAP a great candidate for implementations where reliability and security must be guaranteed and make SOAP very popular in enterprise environments. Private web service within an enterprise network is a great use case for SOAP protocol implementation. Another common use case would be an application using an automated machine to machine communication. With full mapping of objects between client and server (WSDL, XML metadata), SOAP is the natural candidate for these types of applications.

There are situations in which there is no need for full mapping of objects to the client. Sending messages from client to server and vice versa can be expensive in networks where bandwidth is sparse. One obvious use case where this is true are the Web APIs consumed by mobile applications. In this scenario it is wise to avoid SOAP at all costs.[3] REST is much better choice here with its 'Cacheable' constraint[4] which if implemented properly could eliminate some of the communications between the client and server and help to conserve limited bandwidth.

When developing a web service the factor of simplicity could play a role when deciding on the implementation. REST is much simpler then SOAP when it comes to have a prototype up and running as soon as possible. Straightforward JSON to JavaScript object mapping greatly simplifies the payload serialization and development of rich web applications consuming the service. This statement of REST being simpler in implementation than SOAP is only partially true. REST application usually becomes as complex as SOAP later in development when validation and security need to be implemented.[2] The testing must be taken into consideration as well and there is a general consensus that REST APIs are more testable than SOAP services.

Another problem comes with the fact that SOAP protocol is much more restrictive and creates a tight coupling between the client and server. REST APIs are a better choice in open nature of the web. API ease-of-use is commonly more important then contracts between the client and the service. In the world of start-ups, the business requirements change rapidly and APIs must adapt to the with a grace. As mentioned in "SOAP vs REST" section of this paper, changes are much easier to handle in REST architecture. Usage of SOAP in the web environment would be counter-productive because it would bring unnecessary contracts which could stay in the way of the service evolution. The REST seems to be a natural choice for the current web environment.

4 The Rise of Wep APIs

The current Web is changed with the rise of Web APIs over the last 15 years. The Web has become a service-oriented platforms. Services and information are available to internet users through public APIs. You can order groceries from TESCO, watch the video on demand from Netflix or post a tweet. Programmable Web (http://www.programmableweb.com/) is tracking public APIs evolution since 2005.[6] It currently registers 14,819 publicly available APIs.

In 2000 Microsoft released SOAP protocol which allowed applications implemented with different technologies to communicate with each other over the network. A number of companies with leading development platforms such as IBM and Oracle backed Microsoft effort and in a few years, SOAP became a standard for building Service Oriented Architectures.

Meanwhile a new web development platforms gained popularity, namely Ruby on Rails, JavaScript Node.js, Python Django and others offering a great productivity in web applications development. SOAP being designed for enterprise didn't fit very well with these new platforms. It's heavyweight nature and verbosity of XML made work with painful. Developers were looking for something easier to use and turned their attention toward REST architectural pattern and JSON simple serialization format.

Another driving force fuelling rapid growth of REST-like APIs is the Single Page Application (SPA) approach to rich web development. SPA has brought the desktop like application functionality to the web development. Resources are asynchronously requested from the server and dynamically added to the page. The page is never reloaded nor is another page taking the control of the application. Google Mail, Twitter or Facebook are all SPAs. Each of these applications consume REST-full API.

REST is dominant in Web API style, Programmable Web is currently registering 9176 public REST APIs and 2458 public SOAP web services. It's possible that SOAP is dominant in private enterprise but those statistics are not available.

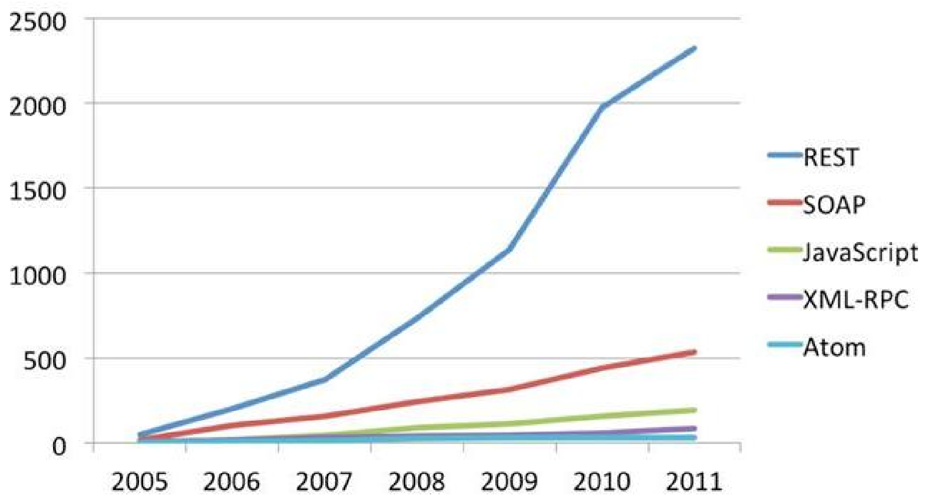

Figure 1: API styles market share in time

4.1 PayPal APIs

PayPal Web API is regarded as an example of well-designed API.[7] PayPal is leading online payment system and is owned by eBay. In this section, I will outline in short what functionality it offers.

Paypal offers three types of interfaces that can be used to incorporate PayPal functionality to a client application, such as a website or mobile app.

- REST API
- SOAP API
- name-value pair NVP API

To addition, the REST Server SDKs (Software Development Kit) are available for Java, PHP, Node.js, Python, Ruby and .NET, plus mobile integration SDK for Android and iOS.

REST API uses HTTPS for transfer, JSON for data serialization and OAuth 2.0 for authorization. API is designed as a REST-full and includes HATEOAS links.

Interesting feature is support for two environments. The sandbox environment is for testing purposes and the Live environment is for production processing. All APIs are very well documented and with technical support available.[8]

5 Conclusion

SOAP and REST are two different approaches to a similar task of designing network based applications. SOAP is a full blown protocol which allows manipulation of remote objects. SOAP with its security, clear contract between the client and server and feature rich XML serialization is a popular choice for private enterprise web service implementations and network applications which require automated machine to machine communications. REST is an architectural style which focuses on operations that could be performed on web resources. REST its operations inherited from HTTP protocol and with its lightweight JSON serialization it is very popular in public Web API development.

References

[1] John Mueller, *Understanding SOAP and REST Basics And Differences*, http://blog.smartbear.com/apis/understanding-soap-and-rest-basics, (2013-01-08)

[2] Armel Nene, *Web Services Architecture – When to Use SOAP vs RESTs*, https://dzone.com/articles/web-services-architecture, (2014-10-16)

[3] Bruno Pedro, *Is REST better than SOAP? Yes, in Some Use Cases*, http://nordicapis.com/rest-better-than-soap-yes-use-cases, (2015-01-15)

[4] Roy T. Fielding, Richard N. Taylor, *Principled design of the modern Web architecture*, https://www.ics.uci.edu/~fielding/pubs/webarch_icse2000.pdf, Information and Computer Science University of California, Irvine Irvine, CA 92697–3425 USA +1.949.824.4121 fielding,taylor@ics.uci.edu (2000)

[5] Mike Rozlog, *REST and SOAP: When Should I Use Each (or Both)?*, http://www.infoq.com/articles/rest-soap-when-to-use-each, (2010-04-10)

[6] Ross Mason, *How REST replaced SOAP on the Web: What it means to you*, http://www.infoq.com/articles/rest-soap, (2011-10-20)

[7] Quora, *What is a good real life example of a truly RESTful API?*, https://www.quora.com/What-is-a-good-real-life-example-of-a-truly-RESTful-API, (2014-12-30)

[8] PayPal, *PayPal Developer*, https://developer.paypal.com/docs/accept-payments/, (2016)