

REST - REpresentational State Transfer

Michael Przybilski

1 Introduction

Many Internet fora, such as newsgroups, web-logs and the like, have lately been fueled by discussions concerning SOAP (Simple Object Access Protocol) vs. REST (REpresentational State Transfer). This report should shed some light into the matter, by attempting to give a comprehensive overview and discussion on the subject of the REST approach.

We will start in Section 2 by giving an overview of the REST architecture approach, stating what REST actually is, how it relates to the Web as it is, as well as how it relates to different standards that govern the Internet today, especially with regard to SOAP and WSDL. In Section 3 a short, practical analysis of REST is provided, while Section 4 discusses the different advantages and disadvantages represented by REST and the issues brought forward by advocated of SOAP and REST. This section will also give an overview of the current discussions of SOAP vs. REST that are currently ongoing. Section 5, together with Section 6 conclude this paper and provide an outlook at the future developments of the REST approach and the coexistence with SOAP.

2 Overview

REST is an acronym of REpresentational State Transfer, a term used by Roy Fielding in his doctoral dissertation, describing an **architecture style** of networked computer systems [2]. REST is thus **NOT** a standard but an approach to developing and providing services on the Internet and is thus also considered an architectural style for large-scale software design.

“Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use.” [2]

The approach described by REST is actually nothing really new. Basically the whole architectural style of the so-called Web follows the REST architecture style. The new issue is that Roy Fielding, one of the main contributors to the development of the HTTP protocol and also otherwise quite an guru with regard to the Internet, has in his doctoral dissertation been able to properly analyze

and structure, what made the Web so successful. The REST architecture style is the result of this analysis.

According to Fielding, REST emphasizes the following citeFielding00:

- the scalability of component interactions,
- the generality of interfaces,
- the independent deployment of components,
- the existence of intermediary components, reducing interaction latency, reinforcing security, and encapsulating legacy systems.

When looking at the Web nowadays, it has certainly achieved most of these goals. The Web has been growing exponentially, while performing always very well. At the same time it enables access with different clients and access mechanisms, thus proving the generality of its interfaces. The "independent deployment of components" is simply a way of dealing with the realities on the Internet, where client and server implementations may be deployed and last for years or even decades, where HTTP servers may never be upgraded, yet newer clients must be able to communicate with them. Older clients on the other hand need to be able to communicate with new server version, without major problems. The needed extensibility for this is in HTTP achieved by having headers, through URI-generation techniques, as well as the ability to create new methods content-types. The "compatibility with intermediaries", such as different kinds of Web proxies to improve performance, or enforce security policies has quite clearly also been achieved. More importantly, these intermediaries enable the encapsulation and integration of non-Web systems into the Web.

The fundamental way how REST achieved these goals is by imposing several basic constraints:

- **Identification of resources** with Uniform Resource Identifiers (URI). The resources that are identified by these URIs are the logical objects that messages are sent to.
- **Manipulation of resources through representations** means that resources are not directly accessed or manipulated, but instead their representations are used.
- **Self-descriptive messages** refers to the fact that HTTP messages should be as self-descriptive as possible in order to enable intermediaries to interpret messages and perform services on behalf of the user. This in turn is achieved by standardizing several HTTP methods (e.g., GET, POST, etc.), many headers and the addressing mechanism. Furthermore, HTTP is by default a stateless protocol allowing the interpretation of each message without any knowledge of the preceding messages.
- **hypermedia as the engine of application state**, enabling the current state of a particular Web application to be kept in one or more hypertext documents, *residing either on the client or the server*. This enables a

server to know about the state of its resources, without having to keep track of the states of individual clients.

As mentioned earlier, REST is an architectural style, not a Standard. Yet it uses many standards, such as

- **HTTP**, the Hypertext Transfer Protocol
- **URL** as the resource identification mechanism
- **XML/HTML/PNG/etc.** as different resource representation formats, or
- **MIME Types**, such as `text/xml`, `text/html`, `image/png`, etc.

As can be seen, also the use of these standards is based on the fundamental characteristics of REST:

- **Client-Server:** a pull-based interaction style: consuming components pull representations.
- **Stateless:** each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server.
- **Cache:** to improve network efficiency responses must be capable of being labeled as cacheable or non-cacheable.
- **Uniform interface:** all resources are accessed with a generic interface (e.g., HTTP GET, POST, PUT, DELETE).
- **Named resources:** the system is comprised of resources which are named using a URL.
- **Interconnected resource representations:** the representations of the resources are interconnected using URLs, thereby enabling a client to progress from one state to another.
- **Layered components:** intermediaries, such as proxy servers, cache servers, gateways, etc, can be inserted between clients and resources to support performance, security, etc.

So-called *RESTful* systems follow the principles of REST, which very much evolves around resources, their addressing and the manipulation of their representation. What is still argued about is whether the distinction between resources and their representations too impractical for the normal use on the web, even though it is popular in the RDF community.

All that applications require are the identifier of the resource, and the action it wishes to invoke. There is no need to know whether there are any intermediaries, such as caching mechanisms, proxies, gateways, firewalls, tunnels, or

anything else between it and the server actually holding the information. Applications still have to be able to understand the format of the information (representation) returned, which is typically an HTML or XML document, depending on the further use, or the presentation to a human. Currently most resources are intended for consumption by humans and thus typically have only one representation, namely HTML. However, this might change in the future, when machine-to-machine communication becomes more important, and when the representation changes based on its further use.

3 Analysis

Adherence to REST will enable the reference of resources available on other machines, using resource identification mechanisms, such as URL. In current discussions, the URL or URI-base is compared to a noun in the human language, where different nouns and their basic or intuitive meaning have been agreed upon. The operations that can be applied on these nouns also need to be agreed upon. This polymorphism, that ability of applying the same verb, or operation to different nouns, or references is also often mentioned in this comparison. The same verb can be applied to different nouns, just like the same operation, such as HTTP's GET can be applied to different resources, such as HTML files, PNG pictures, etc.

While a URL represents the noun the of the human language, the operations, such as GET, POST, etc. represent the verbs that can be applied to them. These basic functionalities are provided for instance by the HTTP protocol, and form according to Fielding the basis of the Web and it's functioning. The problem that is currently addressed in many REST discussions is the agreement of the interpretation of representation of resources and the operations that can be applied accordingly.

Currently most resources are targeted toward consumption by humans and are thus in HTML format. However, since this is not very suitable for the consumption by other machines, there will be different representations of the same resource for different clients, depending on their needs and the further consumption. Ongoing discussions regard the agreement upon, and standardization of universal resource identification mechanisms, or nouns, based on URI and polymorphic operations, or verbs.

The idea is that similar to the field of database management systems, where such an agreement has been established in the form of the Standard Query Language (SQL), also the Web can be base on a very basic structure of universal resource identifiers and polymorphic operations. In SQL the basic resource identifiers can be seen as tables, which are addressed, and the basic operations (e.g., select, insert, update, delete, etc.), which enable the retrieval or manipulation of the representation of the data contained in those tables.

4 Discussion

The following section represents part of the currently ongoing discussions, regarding the Simple Object Access Protocol (SOAP) vs. the REpresentational State Transfer (REST) architecture style.

SOAP and Web Services are a way of bringing distributed systems to a new era, using state-of-the-art technologies and standards, developed by the World Wide Web Consortium (W3C).

“W3C is transforming the architecture of the initial Web (essentially HTML, URIs, and HTTP) into the architecture of tomorrow’s Web, built atop the solid foundation provided by XML.”[4]

While this is a very noble goal, the supporters of the REST approach see several shortcomings. One one them is the Remote Procedure Call (RPC) approach, taken. A REST web application for instance requires a fundamentally different design approach than an RPC application. In RPC, the emphasis is on the diversity of protocol operations, or verbs (see 3). For example, an RPC application might define operations such as the following:

```
getUser()
addUser()
removeUser()
updateUser()
getLocation()
addLocation()
removeLocation()
updateLocation()
listUsers()
listLocations()
findLocation()
findUser()
```

Following the REST approach, the emphasis is on the diversity of resources, or nouns. For example, a REST application might define the following two resource types

```
User {}
Location {}
```

In REST each resource has its own location, identified by its URL. Clients can retrieve representation of of those resources through the standard HTTP operations, such as GET, manipulate it, and upload a changed copy, using the PUT command, or use the DELETE to remove all representations of that resource. Another advantage is that object has its own URL and can easily be cached, copied, and bookmarked. Other operations, such as POST can be used for actions with side-effects, such as placing a purchase order, or adding some data to a collection.

To update for instance a user’s location, a REST client could first download the above XML record using HTTP GET. The client would then modify the file

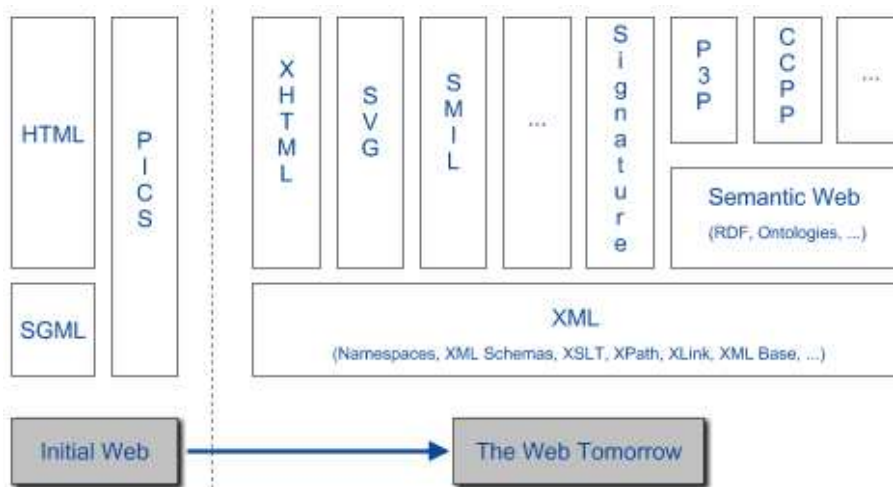


Figure 1: The future Architecture of the Web according to the W3C [4]

to change the location, then upload it again using HTTP PUT.

The "generality of interfaces", as described in Section 2 is what is believed to make it a better basis for a Web Services framework than the SOAP-based technologies. In contrast to SOAP, where all the method names, addressing model and procedural conventions of a particular service must be known, HTTP clients can communicate with any HTTP server, without any further configuration. The reason for this is clear: after all, HTTP is an application protocol whereas SOAP is a protocol framework (and more).

It needs to be noted however, that the HTTP operations do not provide any standard method for resource discovery. Instead, REST data applications work around the problem by treating a collection or set of search results as another type of resource, requiring application designers to know additional URLs or URL patterns for listing or searching each type of resource.

One major difference between the REST point of view and that of the SOAP-based web services architecture is that REST advocates view the Web as an information system in its own right and we advocate that other systems should be integrated into the information system through gateways. [3]

Considering Tim Berners-Lee's vision of the Web:

The Web is simply defined as the universe of global network-accessible information. [1]

Berners-Lee's point of view is that the first goal of the Web is to establish a shared information space. Legacy systems can participate by publishing objects and services into the space. This also reflects in the W3C vision of the future architecture of the Web (see Figure 1).

It is however very different from the terminology that tends to surround Web services. In the Web Services world it is common to talk of the Web as being a "transport" for messages which are essentially only interpreted by systems outside of the Web. In that view, the Web is transport middleware for the systems. This approach is destructive in the long run because the two applications communicating in this way have not established any common ground with all of the other applications on the network. Adding a third application will not be much easier than integrating the first two.

The core of the Web's shared information space idea is the URI. The SOAP-based Web services specifications have never adopted the notion of a Web as a shared information space and thus have not yet fully adopted the Web's model of URI usage. They have rather always presumed that every application would set up its own unique namespace from scratch, instead of using URIs as an addressing mechanism.

The Web Services standards were never designed to use URIs as resource identifiers and in fact WSDL makes this essentially impossible, as it does not enable the expression of a particular element in a SOAP message representing a URI and that the URI points to a web service. Therefore each WSDL describes one and only one Web resource and provides no way to describe links to other resources. SOAP and WSDL use URIs only to address endpoints, which in turn manage all of the objects within them.

One analogy that is often provided is that of a hotel and a wake-up service [3]: A hotel might not have direct dial connections from the outside. In order to call a room, one has to contact the operator first (similar to contacting the "SOAP endpoint") and then ask to connect to a particular room. Now one wishes to buy an outside service, such as a once-a-day automated wake-up call and horoscope service. When trying to sign up for the service, one is asked to enter the phone number to call back, which not not possible, as there is no single number one can provide. The service must contact the operator first and then the operator must patch the service through. Obviously the computer on the other end is not going to be smart enough to know to go through the operator and the operator will not know to patch the call through to ones particular room.

Implementing such a horoscope service would be practically impossible. A particular application of the phone system would simply cease to exist.

The problem with SOAP is that addressing is not sufficiently addressed. Older standards like CORBA and DCOM provided every object with an address and although the address syntaxes were not URIs, they were at least standardized within the domain of each RPC protocol. **SOAP lacks any equivalent addressing model.**

In order to increase flexibility, SOAP does have a binding to HTTP, even though HTTP is recommended as the underlying protocol. That binding inherits the ability to use a URI-based model from HTTP. This binding also inherits the standardized HTTP methods. So there is hope that these SOAP/HTTP services will use URIs in a deep way. At this point it is primarily a question of tools and proper support in the service description languages.

In many ways the standard SOAP model is at odds with the basic ideas of what might be called XML philosophy. The XML family of standards all presume that that all useful information is available in an XML form at a standardized address. Web services technologies are almost always used to set up alternate address spaces and these address spaces are essentially never URI-addressable. Examples of these alternate address spaces include UDDI and Microsoft's .NET My Services. These address spaces are fundamentally incompatible with URI-centric XML technologies.

The loss of RDF and other semantic Web technologies is particularly grievous. These technologies could be used to greatly improve the extensibility of web services and to help them support advanced logical features such as subclassing and inferencing. They could also become the basis for declarative business rules and distributed discovery for web services.

In short, the semantic web technologies have the potential to fill many of the holes in the web services picture. But they can only work with web services that identify resources with URIs. Web services can only identify resources with URIs if the web service toolkits allow them to. Web service toolkits will hopefully follow the lead of the standards and there is hope for further progress on that front.

Web protocols and Web addressing models serve only as "transports" for SOAP. That is like saying that the Web is a taxi driver and its only job is to get SOAP from place to place. The SOAP message can get out of the taxi and into another vehicle for the next leg of its trip. In technical terms, SOAP "tunnels" through the Web. This is not what the Web was designed for, nor what it is good at.

Another issue that is often discussed with regard to SOAP vs. REST is asynchrony, which most commonly refers to the fact that two communicating programs (e.g. a client and server) should not be need to stay in constant communication while one of them is doing a lengthy calculation. There should be some way for one participant to call back to the other. SOAP does not directly support this but it can nevertheless be used in an asynchronous manner by piggy backing on an asynchronous transport. This is not yet standardized but will be one day. There are a variety of approaches to doing asynchrony in HTTP, because there are a variety of aspects to asynchrony. It has been proposed that these should be standardized. One such specification is called "HTTPEvents".

Further issues regard routing and reliability. HTTP messages are already routed from clients to proxies to servers. This is a sort of network-controlled routing. There is also a case to be made that sometimes it makes sense for the client to control routing explicitly by defining a path between nodes. Some REST researchers are experimenting with variants of SOAP Routing for this. Reliability means in most cases the once and only once delivery of a messages. This is relatively easy to achieve using HTTP, but it is not a built-in feature. Arguably, the quest for "guaranteed reliability" is fundamentally at odds with the very nature of large-scale networking and the attempts to achieve it may degrade performance.

Also security is an issue frequently discuss. Although HTTP has security features, it has not been properly compared to the the features that arise as part of WS-Security. This discussion is still very much ongoing, with proposed extensions of HTTPS and the like.

SOAP has an extensibility model that allows a SOAP message creator to be very explicit about whether understanding of a portion of the message is optional or required. It also allows the headers to be targeted at particular intermediaries (e.g. proxies, caches). There is an extension to HTTP with many of the same ideas (and in fact the SOAP version is probably based upon the HTTP version) but it is not as well known nor as syntactically clear.

The issue of service description is frequently discussed as well. While SOAP has WSDL to describe services, HTTP does not provide something similar. Because REST is a document-centric model, REST service description may grow out of schema languages and in particular semantic schema languages like DAML. It may also be possible to use the next version of WSDL in a constrained manner as a type description language for HTTP resources.

One last issue frequently discussed is that of familiarity. The Web Services technologies are designed to be so flexible that they can be used independently of ones programming experience. People with no experience in network programming at all can use the RPC variant and pretend that they are making local function calls. Those with experience using message-oriented middleware may use unidirectional SOAP to wrap their messages. Advocates of REST may use it as an HTTP extension. None of these three different groups of programmers will be able to interoperate with each other, still they will at least be able to say that they are using SOAP. HTTP can only achieve similar levels of familiarity through a completion of the educational process that has been ongoing since the invention of the Web.

5 Conclusions

SOAP, as well as REST have very many advocates. SOAP has the advantage of being heavily supported by the industry, as well as organizations such as the W3C. REST on the other hand has a very good track record and essentially is trying to bring further what has made the Web as successful as it is.

Both approaches have their advantages and disadvantages and it can not even be said that they are opposing each other too much to coexist. One issue that appears to be of great concern is the addressing issue, which seems to be insufficiently addressed in the SOAP approach. This may in the future even create problems with other W3C technologies. For instance it will not be possible to use RDF to assert things about resources that do not have URIs, as it will not be possible to use topic maps and XLinks to build relationships between them. It will also not be possible to use RSS to track changes to the objects, or to use XPointer to point into their XML representations and to use XInclude and XSLT's "document" function to integrate them.

While many systems are using Web Services and SOAP, also explicit REST-based systems are in use, such as

- **Amazon.com** offers its developer interface in both REST¹ and SOAP² versions.
- **eBay** offers a REST developer interface³
- **Yahoo!** offers a REST developer interface⁴

And, as mentioned earlier, essentially the whole Web is based on the REST approach, proving it to be very successful.

6 Future Developments

Currently three different possibilities are seen for the future developments of SOAP and REST. Certainly neither will entirely cease to exist, but it is possible that either will only reside as a nice-technology. It is possible, that the critique of the SOAP philosophy, which REST advocates currently bring forward is unfounded. In such a case it is likely that the SOAP model turns out to be good enough and REST's use will be restricted to the interactive hypertext Web. If however the flaws in SOAP's methodology's are gradually revealed to be deep and intractable, REST or something like it will become the dominant architecture and SOAP is relegated to niches or to a minor supporting role. One possibility is however also that REST and SOAP find ways to not only coexist, but to work together, either focusing on different problem domains or interoperate so well that it becomes easy to use the two at the same time.

References

- [1] Tim Berners-Lee. The world wide web: Past, present and future. <http://www.w3.org/People/Berners-Lee/1996/ppf.html>.
- [2] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.
- [3] Paul Prescod. Roots of the rest/soap debate. http://www.prescod.net/rest/rest_vs_soap_overview/.
- [4] World Wide Web Consortium (W3C). About the world wide web consortium (w3c). <http://www.w3.org/Consortium/>.

¹<http://www.amazon.com/gp/aws/landing.html>

²<http://en.wikipedia.org/wiki/SOAP>

³<http://developer.ebay.com/rest/>

⁴<http://developer.yahoo.com/>