REST vs SOAP
by
Chris Venour

What are REST and SOAP?

REST and SOAP are web services that differ in the way they get a client and a web server to exchange information over the Web. Three technologies form the core of the Web: (1) the Uniform Resource Identifier (URI) naming standard, (2) the HTTP protocol and (3) HTML (hypermedia). REST and SOAP differ in the extent to which they use these three fundamental technologies on which the Web is built.

How REST works

REST uses the three core web technologies mentioned above in the following way:

1. REST assigns a unique URI to every resource. (A resource is just a named piece of data. Things that are important for a given application are called resources).

2. REST uses the methods of the HTTP protocol to operate on resources. Some of these HTTP methods are GET, PUT, POST, DELETE, and HEAD. For instance, to create a resource, a client sends a PUT request to a URI. The URI contains the desired name of the resource and the REST service creates a resource with that name. To delete a resource, a client sends a DELETE request to that object's URI [1].

3. REST can implement a hypermedia model the way HTML does. In other words, REST can link resources together [3].

REST doesn't re-invent the wheel. In order to exchange information over the Web, it makes use of a technology that already exists: the technology that undergirds the Web itself. Using the Web's technology confers a lot of benefits. For instance, there are often many HTTP intermediaries such as proxies and gateways between a client and a webserver, and these intermediaries can improve the transfer of data [2]. For one thing, these intermediaries can cache data so that a client's request doesn't have to reach all the way to the webserver. However, caching can happen only if the HTTP request is a GET. REST services make GET requests but as we'll see in the next section, SOAP only uses the POST method and so misses out on the efficiency and scalability that caching provides. The use of gateways also allows traffic to be distributed among a large set of webservers based on the different HTTP methods used or on the different URIs, and this also increases scalability [2]. These are just a couple of ways that SOAP loses out by not making use of the web technologies the way REST services do.

How SOAP works

SOAP makes much less use of the three core web technologies: it "reinvent(s) or ignore(s) every feature that makes the Web successful" [1]. For this reason, people sometimes say that a SOAP service is built "on top" of the Web and works against it whereas a REST service is "inside" or "part of" the Web and works with it [1][2]. Specifically, a SOAP service uses the web's technologies in a limited way because it:

1. uses only one URI. In other words, a single URI is "overloaded to support multiple operations" [1].

2. uses only one of the HTTP methods: the POST method. For instance, if a SOAP service wants to get information or delete information from a web server, it doesn't use the HTTP methods GET or DELETE but uses the POST method for both of these very different requests.

3. is not well-connected i.e. does not link resources.

In other words, a “typical SOAP service ... looks a lot like a C library. There are a bunch of functions, sometimes namespaced with periods. All of these functions are accessed by sending a POST request to one single URI” [4]. The consequence is that SOAP web services are "not addressable, cacheable or well connected, and they don't respect any uniform interface ... They're opaque and understanding one doesn't help you understand the next one ... (and) they also tend to have interoperability problems when serving a variety of clients" [1].

When should you choose REST?

REST should be your default choice when implementing a web service because it uses the core web technologies much more than SOAP does. That means REST is faster, "simpler, more scalable, easier to use ... and better able to handle a wide variety of clients than are services based on SOAP" [1]. SOAP is more complicated and more computationally expensive but there are some situations where you might want to choose SOAP over REST. These situations are briefly mentioned in the next section.

When should you choose SOAP?

Despite its complexity and performance overhead, there are some situations when you might want to choose SOAP over REST. For instance, when your service:

1. **requires high security**: "security concepts are much better specified and deployed in SOAP-based protocols" than in REST [1]. This is thanks to some technologies that have been built on top of SOAP. These technologies are often referred to as the WS-* stack and they provide functionality which REST doesn't natively possess. For instance, WS-Security[1] can encrypt messages, detect whether a message has been tampered with and "prove who the originator of (the) ... message was long after it was sent" [1]. For this reason, SOAP is used more often than REST is in the financial services sector [3].

2. **performs complicated transactions**: transactions such as "transfer $50 from bank A to bank B" are "insanely difficult to implement, particularly in a distributed environment" and the WS-AtomicTransaction and WS-BusinessActivity standards that SOAP provides can help with this task [1].

3. **requires non-HTTP transports**: SOAP messages are "independent of the protocol used to transport the message". This means that, unlike a REST request, a SOAP message can be sent on non-HTTP transport protocols such as FTP/JMS/MQ/SMTP etc. [3].

---

[1] "SOAP+WS-Security is like a suit of armor. If you don't need a suit of armor, it's heavy and it's clunky. But if someone is coming at you with a sword, it's kind of useful to be wearing it" [3].

Note that IDEs like VisualStudio, Eclipse, and NetBeans can automatically generate SOAP services that use the WS-* stack and so implementing a highly secure web service or a complicated transaction service is easier using SOAP than it would be using REST.

References

1. Leonard Richardson and Sam Ruby (2008), *RESTful web services* (book)
2. Joe Gregorio *Intro to REST* (2009) (Google Developers video)
3. Tom Burnell *REST, SOAP, Hypermedia: When, why & how to use what* (2013) (talk at Nordic APIs conference)
4. Leonard Richardson *The Maturity Heuristic* (2008) (blog) https://www.crummy.com/writing/speaking/2008-QCon/act3.html
5. Jon Flanders RESTful.NET (2009) (book)