<https://stackoverflow.com/questions/19884295/soap-vs-rest-differences>

SOAP and REST can't be compared directly, since the first is a protocol (or at least tries to be) and the second is an architectural style. This is probably one of the sources of confusion around it, since people tend to call REST any HTTP API that isn't SOAP.

Pushing things a little and trying to establish a comparison, the main difference between SOAP and REST is the degree of coupling between client and server implementations. A SOAP client works like a custom desktop application, tightly coupled to the server. There's a rigid contract between client and server, and everything is expected to break if either side changes anything. You need constant updates following any change, but it's easier to ascertain if the contract is being followed.

A REST client is more like a browser. It's a generic client that knows how to use a protocol and standardized methods, and an application has to fit inside that. You don't violate the protocol standards by creating extra methods, you leverage on the standard methods and create the actions with them on your media type. If done right, there's less coupling, and changes can be dealt with more gracefully. A client is supposed to enter a REST service with zero knowledge of the API, except for the entry point and the media type. In SOAP, the client needs previous knowledge on everything it will be using, or it won't even begin the interaction. Additionally, a REST client can be extended by code-on-demand supplied by the server itself, the classical example being JavaScript code used to drive the interaction with another service on the client-side.

I think these are the crucial points to understand what REST is about, and how it differs from SOAP:

* REST is protocol independent. It's not coupled to HTTP. Pretty much like you can follow an ftp link on a website, a REST application can use any protocol for which there is a standardized URI scheme.
* REST is not a mapping of CRUD to HTTP methods. Read [this](https://stackoverflow.com/questions/19843480/s3-rest-api-and-post-method/19844272#19844272) answer for a detailed explanation on that.
* REST is as standardized as the parts you're using. Security and authentication in HTTP are standardized, so that's what you use when doing REST over HTTP.
* REST is not REST without [hypermedia](https://stackoverflow.com/a/29586455/1202421) and [HATEOAS](http://en.wikipedia.org/wiki/HATEOAS). This means that a client only knows the entry point URI and the resources are supposed to return links the client should follow. Those fancy documentation generators that give URI patterns for everything you can do in a REST API miss the point completely. They are not only documenting something that's supposed to be following the standard, but when you do that, you're coupling the client to one particular moment in the evolution of the API, and any changes on the API have to be documented and applied, or it will break.
* REST is the architectural style of the web itself. When you enter Stack Overflow, you know what a User, a Question and an Answer are, you know the media types, and the website provides you with the links to them. A REST API has to do the same. If we designed the web the way people think REST should be done, instead of having a home page with links to Questions and Answers, we'd have a static documentation explaining that in order to view a question, you have to take the URI stackoverflow.com/questions/<id>, replace id with the Question.id and paste that on your browser. That's nonsense, but that's what many people think REST is.

This last point can't be emphasized enough. If your clients are building URIs from templates in documentation and not getting links in the resource representations, that's not REST. Roy Fielding, the author of REST, made it clear on this blog post: [REST APIs must be hypertext-driven](http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven).

With the above in mind, you'll realize that while REST might not be restricted to XML, to do it correctly with any other format you'll have to design and standardize some format for your links. Hyperlinks are standard in XML, but not in JSON. There are draft standards for JSON, like [HAL](http://stateless.co/hal_specification.html).

Finally, REST isn't for everyone, and a proof of that is how most people solve their problems very well with the HTTP APIs they mistakenly called REST and never venture beyond that. REST is hard to do sometimes, especially in the beginning, but it pays over time with easier evolution on the server side, and client's resilience to changes. If you need something done quickly and easily, don't bother about getting REST right. It's probably not what you're looking for. If you need something that will have to stay online for years or even decades, then REST is for you.