Introduction to Web Services

"Simplicity is prerequisite for reliability" (E.W. Dijkstra)

Why Web Services?

Software applications employ a variety of disparate formats to store and exchange data in dissimilar ways and therefore cannot "talk" to one another productively. Web services are a practical, cost-effective solution to this communication problem suffered by critical applications running over different operating systems, platforms, and using different language that were previously impassable. Web services are hardware, programming language, and operating system independent. This means that applications written in different programming languages and running on different platforms can seamlessly exchange data over intranets or the Internet using Web services.

A web service is a self-contained and self-describing web application that is available over the Internet or an intranet and that uses open standardized messaging protocols, generally XML-based, to exchange data directly with other web applications. Web services are designed to be accessed by other applications. Web services have two main applications:

- ✓ To provide a fixed, embeddable and reusable data commonly needed very frequently by other services, which use the data to empower their own web applications. For example you could use small web services providing weather reports, bus timetables and news to source your own web application. These types of web services are also known as widgets. They normally implement things that are needed very often, saving the time of doing them over and over again.

- ✓ To connect existing web applications; by giving those applications a way to share their data and make them interoperable. Thus the web service of *Twitter*, the most popular micro-blogging service on the Social Web, enables other applications to use the user's tweets to create other services. For example, the *Tums!* web service allows users to inspect *Twitter*-based profiles and enables other web applications to re-use these profiles to create more additional services for example to allow users to overview their personal Twitter activities or to explore the topics those users were concerned with in the past.

Therefore, the benefits of using Web Services include

(1) allowing you to expose the functionality of your existing code over the network so other applications can use the functionality of your program;
(2) interoperability, which means allowing different applications to talk to each other and share data and services among themselves (making the application platform and technology independent);
(3) using open standardized protocols over HTTP for the communication and;
(4) allowing you to use your existing low cost internet for implementing Web Services.

The Web Services Vision

Support distributed applications composed of independent processes which communicate by message passing. Targets loosely coupled systems. Enables communication between web services implemented in different languages, and by different companies, and on different platforms

Type of Web Services

Web services are of two kinds: Simple Object Access Protocol (SOAP) and Representational State Transfer (REST).

SOAP uses all the four layers (Service Transport, XML Messaging, Service Description and Service Discovery layers) defined by the W3 protocol for Web Services. While SOAP is the prevailing standard for complex web services, and hence has better support from other standards such as WSDL and WS-* (extensibility) and has built-in error handling tools, it can sometimes be slower than middleware technologies like CORBA or ICE due to its verbose XML format. SOAP is also found to be more verbose, conceptually more difficult to understand, more "heavy-weight" to implement. Due those reasons the use of SOAP in web services is gradually decreasing.

The origin of the term "REST" comes from the famous thesis from Roy Fielding describing the concept of *Representative State Transfer* (REST). REST is much simpler than SOAP. REST does not contain messaging layers or additional protocols and standards. Instead, it focuses on using the standard four HTTP operations for creating stateless services. An application can simply access the resource using the unique URI and a representation of the resource. With each new resource representation, the client is said to transfer state. The emergence of the RESTful style of web services was a reaction to the over demanding SOAP-based standards. In RESTful web services, the emphasis is on simple point-to-point communication over HTTP using plain old XML.

RESTful Architectural Style

As noticed, REST is a simple way of sending and receiving data between client and server and it doesn't have very many standards defined. You can send and receive data as XML or even JSON or plain text. It is light weighted compared to SOAP.

Critics of non-RESTful web services often complain that they are too complex and based upon large software vendors or integrators, rather than typical open source implementations. This, however, has changed since WSDL 2.0 defines a full complement of non-SOAP bindings (all the HTTP methods, not just GET and POST) and the emergence of WADL as an alternative to WSDL. The W3C also recognises REST as one of the major classes of web services and states that the primary purpose of REST-compliant Web services is to manipulate XML representations of Web resources using a uniform set of "stateless" operations.

While accessing RESTful resources with HTTP protocol, the URL of the resource serves as the resource identifier and GET, PUT, DELETE, POST and HEAD are the standard HTTP operations to be performed on that resource.

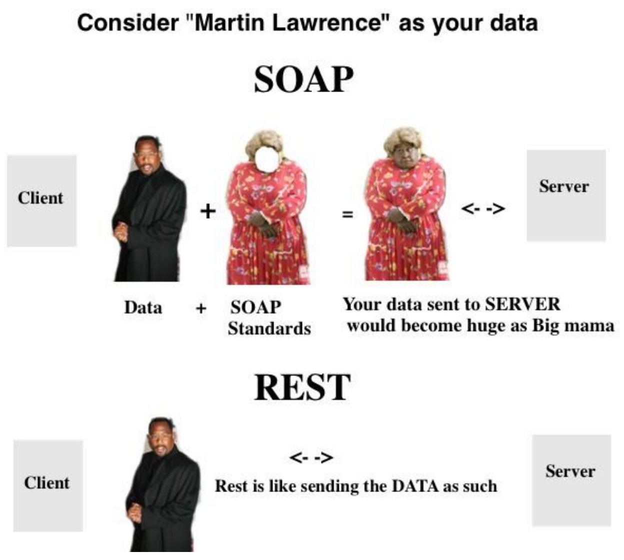
Rest - Representational state transfer

REST is an architectural style that can be summed up as four verbs (GET, POST, PUT, and DELETE from HTTP 1.1) and the nouns, which are the resources available on the network (referenced in the URI). The verbs have the following operational equivalents:

```
HTTP      CRUD Equivalent
==============================
GET       read
POST      create,update,delete
PUT       create,update
DELETE    delete
```

A service to get the details of a user called 'dsmith', for example, would be handled using an HTTP GET to http://example.org/users/dsmith. Deleting the user would use an HTTP DELETE, and creating a new one would mostly likely be done with a POST. The need to reference other resources would be handled using hyperlinks and separate HTTP request-responses.

SOAP (XML) vs REST (JSON)

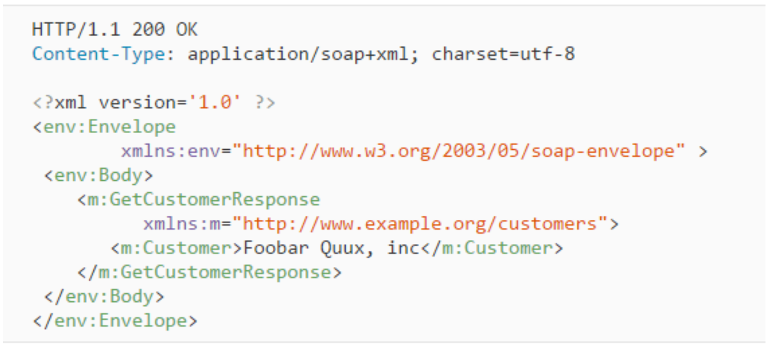

Consider "Martin Lawrence" as your data
SOAP
Client
+
=
<- ->
Server
Data
+
SOAP Standards
Your data sent to SERVER would become huge as Big mama
REST
Client
<- ->
Rest is like sending the DATA as such
Server

First, when using SOAP to request a specific customer record, the request will be an HTTP POST and look like this:

```
POST /customers HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8

<?xml version="1.0"?>
<soap:Envelope
          xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Body>
    <m:GetCustomer
          xmlns:m="http://www.example.org/customers">
      <m:CustomerId>43456</m:CustomerId>
    </m:GetCustomer>
  </soap:Body>
</soap:Envelope>
```

After looking up the customer, the response will look like this:



```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8

<?xml version='1.0' ?>
<env:Envelope
        xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
 <env:Body>
    <m:GetCustomerResponse
          xmlns:m="http://www.example.org/customers">
       <m:Customer>Foobar Quux, inc</m:Customer>
    </m:GetCustomerResponse>
 </env:Body>
</env:Envelope>
```

Now, contrast that to using REST over HTTP with JSON as the exchange data format. The request for the customer record is an HTTP GET:

```
GET /customers/43456 HTTP/1.1 Host: www.example.org
```

And the response:

```
HTTP/1.1 200 OK Content-Type: application/json; charset=utf-8
{‘Customer’: ‘Foobar Quux, inc’}
```

Both of these calls are equivalent. The SOAP version sends a lot more data in both the request and response. Clearly, REST is a simpler way of sending and receiving data between client and server and it doesn't have very many standards defined. You can send and receive data as JSON, XML or even plain text. It is light weighted compared to SOAP. REST typically uses normal HTTP methods instead of a big XML format describing everything. For example to obtain a resource you use HTTP GET, to put a resource on the server you use HTTP PUT. To delete a resource on the server you use HTTP DELETE. REST typically is best used with Resource Oriented Architecture (ROA). In this mode of thinking everything is a resource, and you would operate on these resources. As long as your programming language has an HTTP library, and most do, you can consume a REST HTTP protocol very easily.

Further Reading

RESTful Web Services by Leonard Richardson and Sam Ruby
Professional PHP Web Services by James Fuller, Harry Fuecks
http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
http://coreymaynard.com/blog/creating-a-restful-api-with-php/
http://keithba.net/simplicity-and-utility-or-why-soap-lost
http://searchsoa.techtarget.com/tip/REST-vs-SOAP-How-to-choose-the-best-Web-service
"Web services, service-oriented architectures, and cloud computing the savvy manager's guide" by D. K. Barry et al. (2013) - Full-text available in HWU library.