White Paper

Representational State Transfer (REST)

a) Abstract:

REST is a framework built on the principle of today's World Wide Web. Yes it uses the principles of WWW in way it is a challenge to lay down a new architecture that is already widely deployed and same time makes sure that it won't adversely impact or destroy the very principles on which WWW was Architecture and built and needless to say which laid path for WWW to succeed.

WWW principles are simple and stubborn in way it withhold lot of architectures built on this from many years for example Web Services, which uses the simple WWW principles by piggy backing their own architecture on WWW.

This paper discusses the way REST is implemented and the difference between a traditional Web Services vs. Rest based Web Services, and also gives a oversight of the architecture and need for this in the real world. This paper eventually will enable user to consider REST as one of the possible choices while designing solutions for the Web.

b) Table of Contents

Contents

c) What is REST?

Representational state transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web. The terms “Representational State Transfer” and “REST” were introduced and defined in 2000 by the doctoral dissertation of Roy Fielding.[1][2]. One of the principal authors of the Hypertext Transfer Protocol (HTTP) specification versions 1.0 and 1.1. Systems which follow REST principles are often referred to as “RESTful” (Excerpts from Wikipedia).

REST is an architectural style that treats the Web as a resource-centric application. Practically, this means each URL in a RESTful application represents a resource.

Traditional Web Applications access resources using HTTP GET or POST operations. In Contrast, RESTful applications access resources following the create, read, update, and delete (CRUD) style using the full range of HTTP verbs (POST, GET, PUT and DELETE). As HTTP is a stateless so is REST.

I will take a small example and explain how a typical REST URI and traditional Web application URI looks like. I am having set of services which can return reason codes and I wanted to provide a url where end users can query and see what this reason code is.

Typical Web Application

http://<url>/<webcontext>/getReasonCodeInfo.do?rsncode =305. Here I am using a generic URI for all reason codes, whereas REST every URI is a specific to a specific resource.

http://<uri>/<webcontext>/reasoncode/305 this particular URI is nothing but just sending a http GET request for resource “/reasoncode/305”.

d) RESTful Service Design.

Rest is no way replacement for complex SOA architecture, but for a simple SOA where you want to use HTTP given advantages REST is the best choice.

Today you can see this REST simply implemented in Twitter and very complex structure in Amazon. Readers can further browse through discussion on how Amazon, Twitter and Yahoo implemented it. Typical SOA architecture design starts by identifying the things like

1. Identifies the service provider

   Service can be as simple as weather report and provider can be simple mathematical engine.

2. Decide on format in which data transfer to happen, and this data format can range from simplest to complex structures and types.

   Data type that is interchanged here can be audio, video feeds to XM's and once the type is decided then comes the format, for simplicity we will talk about XML. We know XML needs to have schema defined. So in this case schema can be very complex or a simple format.\

3. Mode of Transport

   I can access service via different transports like JMS, WebServices or HTTP.

Now let's talk about the ingredients required for REST architecture.

1. Decide on the resources and their descriptive URL's

   As we talked about REST is all about resource, so based on the application decide the structure. For example here is the simple weather based service

   http://weather.com/region/USA

   http://weather.com/zipcode/94583

2. Choose a data format for communication on each URL

   Since REST is using HTTP as underlying transport we can return response in a format of plain HTTP response codes to complex XML or JSON string.

3. Specify the methods on each resource.

   In case of CRUD operations we can assign respective http actions for resource.

   For Create we can use POST operation Delete = DELETE , Read = GET

4. Specify the returned data and status codes.

   Since this is a service oriented architecture there needs to be understanding with user on the returned data and status codes.

With these simple steps it's easy to implement REST design.

e) REST VS WebServices

The main advantages of REST web services
are Light weight – not a lot of extra XML markup
Human readable results
Easy to build – no toolkits required
SOAP also has some advantages
Easy to consume – sometimes
Rigid- type checking, adheres to a
contract Development tools

f) REST Advantages

With REST resource centric architectures each resource having its own URL's combine the simplicity of something easy to remember with the capability of reaching the billions of available web pages.

Another benefit of the RESTful interface is that requests and responses can be short. SOAP requires an XML wrapper around every request and response. Once namespaces and typing are declared, a four or five digit stock quote in a SOAP response could require more than 10 times as many bytes as would the same response in REST.

Since HTTP bases/ REST-ful API's can be consumed using simple GET requests, intermediate proxy servers/ reverse-proxies can cache their response very easily. On the other hand, SOAP requests use POST and require a complex XML request to be created with makes response caching difficult.

g) Implementation Techniques

Since it is on HTTP, there are different ways we can implement this from different programming techniques and languages. For example it can be done on either Java, Ruby, Python, C#, Java Script or PHP.

In this paper I will talk about Java implementation.

REST implementations ranges from simple Java Servlet to specialized Restlet(Open Source Implementation).

Today all commercially available and open source Application Servers and Web Technologies stack supports REST. Prominent addition is Spring 3.0 has included REST support.

WebServices testing tools that are available today in market also supports REST web services testing (Eg: SOAP UI 3.0).

I am depicting a small application where I have done REST via plain Servlets. I have and series of services which can return reason codes from the set of hundreds. To provide a comprehensive list to end users I used a simple REST where every reason code will be a resource. My URL looks something like this.

http://<host>/<webcontext>/reasoncode/305

This particular one is trying to bring all the information that we can provide to end user about reason code, and below is my implementation of this. Here I haven't talked about the format I am returning the information, since I have response with me, I can send any type I want, either a plain HTML page or a XML or JSON structure. Creativity/Usability has no limits here.

```
protected void doPost(HttpServletRequest request,
HttpServletResponse response)throws
ServletException, IOException
{
  Pattern pattern = Pattern.compile("^/?.*?/reasoncode/$");
  Matcher matcher = pattern.matcher(request.getRequestURI());
  String reasonCodeInfo = null;
  if(matcher.matches())
  {
  String reasoncode = matcher.group(1);
  reasonCodeInfo = rsnCodeService.getReasonCodeInfo(reasoncode);
  }
  response.getWriter().write("<b>reasonCodeInfo: </b>"+reasonCodeInfo);
}
```

h) Conclusion

In this paper you learned about REST and how WSDL 2.0 REST Web Services use HTTP and XML for communication. RESTful applications are resource-centric as opposed to action-centric.

Although REST doesn't have an exact specification for how to implement it, out-of-the-box support for REST is increasing. Instead of following standards, you need to follow some conventions.

As your application scales it is likely that you will abstract away from the REST implementation details more and more, after a certain point of time heavy machinery might turn out to be cost effective than initial lightweight technology.

In the end I believe SOAP isn't that simple, it requires greater implementation effort and understanding on the client side while HTTP based or REST based API's require greater implementation effort on the server side. API adoption can increase considerably if a HTTP based interface is provided. In fact, an HTTP based API with XML/JSON responses represents the best of both breeds and is easy to implement on the server as well as easy to consume from a client.

For consuming web services, it's sometimes a tossup between which is easier. For instance Google's AdWords web service is really hard to consume, it uses SOAP headers, and a number of other things that make it kind of difficult. On the converse,Amazon's REST web service can sometimes be tricky to parse between it can highly nested, and the result schema can vary quite a bit based on what you search for.

Which every architecture you choose make sure it's easy for developers to access it, and well documented. In the end when you host WebService for the internet, it's the client side complexity that matters most in attracting them to use your service. Choose wisely.

i) References

http://en.wikipedia.org/wiki/Representational_State_Transfer
http://rest.elkstein.org/