Representational State Transfer as a Web Service

A Master's Project

Presented to

Department of Computer and Information Science

State University of New York

Polytechnic University

In Partial Fulfillment

Of the Requirements of the

Master of Science Degree

By

Dhruv Desai

Dec 2015

Representational State Transfer as a Web Service

Except where reference is made to the work of others, the work described here is my own or was done in collaboration with my advisor and/or the members of the advisory committee. Further, the content of this work is truthful in regards to my own work and the portrayal of other's work. This work, I claim, does not include proprietary or classified information to the best of my knowledge.

Dhruv Desai

Representational State Transfer as a Web Service

Master of Science Project in Computer and Information Sciences
Department of Computer Sciences
SUNY Polytechnic Institute

Approved and recommended for acceptance as a project in partial fulfilment of the requirements for the degree of **Master of Science in Computer and Information Sciences**

DATE

Saumendra Sengupta, Ph.D (Adviser)

Jorge Novillo, Ph.D.

Bruno Andriamanalimanana, Ph.D.

Acknowledgment

I, Dhruv Desai want to express my gratitude to Dr. Saumendra Sengupta, the advisor of this project has been a constant inspiration and has offered guidance in every possible way one could imagine. I would also like to thank my family and friends for their encouragement.

Abstract

This report is a study on Representational State Transfer architectural style and its usefulness for implementing web service. This report will highlight the differences in perceiving REST as an architectural style and as a web service. This document will also discuss web services in general and highlight important differences between the different web services in programming languages. The goal of this report is to clarify the term REST as an architectural style which has proved to be a popular choice for implementing a web service rather than REST being termed as a web service and compare Web Services based on its performance in a Java Application.

Keywords: RESTful Web services, SOAP web service, Performance Evaluation of REST

Contents

Chapter 1
Introduction

For more than a decade now, Web has been the ultimate source of information, continuously increasing its widespread use, evolving in speed and content processing. The underlying Web Technologies and Web services being used to process this enormous source of information has evolved as well. This evolving Web, that is being used today, was named Web 2.0 and is built upon features such as Human-Computer Interaction, collaboration among applications and its users and offers a set of services, which in turn are accessible by other users to reach their own goals. These set of services, also known as Web Services has been defined by various authors, by more or less using the same protocols and languages as is evident by the following definition, " Web Services are software components that are developed using specific technologies from three primary technology categories; an XML-based description format (for example, WSDL), and application messaging protocol (for example, SOAP), and a collection of transport protocol (for example, HTTP)(Adams et.al., 2002)" or as defined by "A Web Service is a software component described via WSDL and capable of being accessed via standard network protocols such as but not limited to SOAP over HTTP(Broberg, 2002)".

From both the definitions above, we understand that WSDL(Web Services Descriptive Language) is the base on which the original Web services were specified. The figure below describes how WSDL is used in web services specification. Service Consumer is on the right side of the figure and Service Provider is on the left side.

The procedure of providing a service and consuming it involves various steps. First the Service provider describes the services it provides using WSDL. This service description is published to services repository. The repositories themselves could be using Universal Description, Discovery and Integration(UDDI) or any other form of directory. Next, the Service Consumer issues its requirements to a repository. The repository, as a response, sends out a part of WSDL to the service provider. This part of WSDL explains the consumer what are the

requests and responses for the concerned service provider. Only then can the service consumer send a request to the service provider and the service provider responds with the appropriate information required by the consumer.

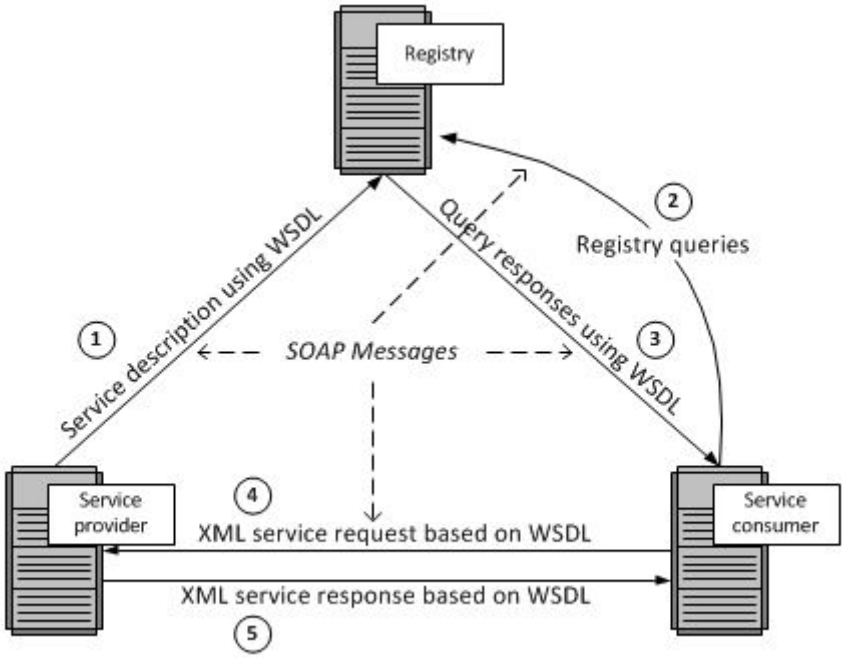


Fig 1.1 Use of WSDL (source: service-architecture.com)

When we hear the term Web Service; SOAP based Web Service and RESTful Web Services are the two kinds of web services that comes to mind. Whereas Simple Object Access Protocol(SOAP) is based on the concept of distribution using RPC mechanism and packaging it in a format using XML and HTTP, Representational State Transfer(REST) based Web services is an architectural style of software which emulates HTTP and similar protocols and constraints them to a defined set of generic operations. SOAP based web services and RESTful web services are used to develop various applications such as web applications, conferencing, etc.

Web Services allow information exchange and platform-independent communication within web applications. This is one of the most promising feature for reusing distributed systems, which increases the value of software assets that exists. Web Services gives power to any application on the internet to potentially reach any other web application. If Web applications can communicate on the internet in compliance with the web service standards, then they can exchange any data or informational messages over internet independent of processors, operating systems, internet protocols, server and programming languages.

In this paper, I strive to survey REST as it was defined as an Architectural Style and its journey towards becoming a popular web service architecture. Other Web services such as SOAP based web service and XML-RPC based web service will be described in brief and then a detailed comparison is drawn out between REST based web service and SOAP based web services. I have also pointed out the benefits of REST over SOAP and how the web service developers are getting more and more inclined towards the use of REST based web service because of its simplicity and efficiency.

I have also developed a Java based application to showcase RESTful web services and its various requests and response. In this application, the server gets request from the client in JSON parsed format and the response in sent out back in the same format. For Client Side, I have used 'Postman', which is a widely accepted Google web app. For Server Side, Jersey framework was used to develop the Server Application which is based on JAX-RS. This application focuses on four requests, namely, 'GET', 'POST', 'UPDATE' and 'DELETE'. A detailed explanation of the working of the application will be explained in the latter part of the report.

Chapter 2

Representational State Transfer as an Architectural style

Representational State Transfer (REST) was first defined as an architectural style specifically for distributed hypermedia systems. It lays down a defined set of architectural constraints that emphasizes on the scalability of component interactions, interface generality, independent deployment of components and intermediary components to reduce interaction latency, encapsulate legacy systems and enforce security (Fielding, 2000). The criterion applied to REST are not bound to any specific set of rules or protocols.

REST is determined as a hybrid architectural style which is derived from several other network-based styles of architecture and it is combined with other additional constraints that defines the uniform connector interface. The REST architectural style is described using six constraints. All of these constraints, when applied on the architectural style defines the basis of RESTful style. Each of the six constraints can be described as below:

A) Uniform Interface

This constraint is the definition of the interface between the server and client. It decouples and simplifies the architecture, which in turn, enables each part of client and server to evolve on its own. The four principles that guides the uniform interface are:

1) Resource-Based

Resource Identifiers are the individual resources that can be identified in requests as part of URI's. The Resources are themselves separate from the conceptual representations that client returned. For Example, the server cannot transfer the whole of database, instead, some XML, JSON or HTML that represents and expresses the database records in user friendly format, for instance, in UTF-8. It all depends on the request details and the implementation on server.

2) Manipulation of Resources through Representations

   Whenever a client request has the representation of a resource, including any attached metadata, if present; it has more than enough information to delete or alter or modify the server resource, provided it has the permission to do so.

3) Self-descriptive Messages

   Each and every message from client has enough information in it that can describe the processing of that message. For Example, information such as which parser to invoke may also be described by an Internet media type (which was known as MIME type previously). The responses sent back by the server also distinctively mention their cache-ability.

4) Hypermedia as the Engine of Application State(HATEOAS)

   Clients distribute state via contents of body, request headers, query-string parameters and the resource name (or the requested URI). The Server delivers its state back to clients via response codes, response headers and body content. This is usually referred to as hyperlinks within hypertext (hypermedia).
   HATEOAS is also referred to as links that are described in the returned header or body that retrieves the object or related objects by supplying the URI.

The Uniform Interface is fundamental to its design provided by any REST service.

B) Stateless

Statelessness is a key in REST architecture as REST is an acronym for Representational State Transfer. It essentially means that the required state that handles the request is imbibed in the request itself, either as a part of the header, body, query-string parameters or the URI. The body has the state (or change of state) of a resource and the URI differentiates that resource. Once the server is done with its processing, the respective state is sent back to the client as a response.

REST asks its users (client) to include all the information required by the server to fulfil that request, resending the state, if that state requires to span in multiple

requests. Statelessness helps in greater scalability as the server is not required to maintain or communicate the state to its session. Also, Load balancers need not worry about session for stateless systems.

C) Cacheable

As on the World Wide Web, clients (browsers) have the ability to cache responses. It becomes essential on the part of the response, to implicitly or explicitly, mention themselves as cacheable or non-cacheable. This allows in preventing clients from using inappropriate or stale data as a response to future requests. Well-managed caching can eliminate client-server interactions to some extent. It can further improve performance and scalability.

D) Client-Server

Servers are separated from clients by the Uniform Interface. This separation of planes benefits both and clients and servers in its own ways. For Clients, it means that they do not have to be concerned about data storage, which is internal to each server, which in turn improves the portability of client code. For Servers, it means that they do not have to be concerned with the User state or the User Interface, which allows the servers to be more simple and scalable. Also, as long as the uniform interface between the client and server is not modified, clients and servers may also be developed independently or replaced completely.

E) Layered System

Because of the Layered system, a client can never be sure whether it is directly connected to an end server or a server intermediary along the way. This enables system scalability, load balancing and provides shared caches. Layering can also enforce security policies.

F) Code on Demand

Servers have the ability to customize or extend a client functionality, albeit temporarily, by transferring the business logic to the client so that it can execute. For

instance, Java applets and JavaScript, which is a client-side script which are compiled components.

Complying with all the constraints described above will conform as a REST architecture style. It enables all kinds of distributed hypermedia system to have various properties, such as, scalability, simplicity, performance, visibility, modifiability, reliability and portability.

Chapter 3

Representational State Transfer as a Web Service

Richardson and Ruby (2007) coined the term RESTful Web Services, and classified the web services that comply with the criteria led down by the REST architectural style. Along with this, these authors brought REST and HTTP together, which gave a technological aspect to the defined abstract criteria. Shortly after REST was defined, W3C (World Wide Web Consortium) revised HTTP as default protocol for Web to transfer messages. It meant that system designed with respect to the RESTful paradigm had a potential audience base of all the devices that are connected to World Wide Web, which implies an infinite number. So, RESTful Web Services can be called portable by using ready-to-use infrastructure of Web and with its benefits, there will not be any dependency whatsoever in terms of software or hardware.

RESTful approach comprises of five concepts, which are, Resource, Representation, Unified Interface, uniform identifier and execution scope and three principles, which are, statelessness, addressability and connectedness. RESTful services work as any other web service, as in, it receives a request that calls the procedure to execute, and it returns a response that contains the result of the execution. However, there are two main differences that can be observed on requests that specifically belong to RESTful services. First, the manner in which the action is defined that will be executed and Second, the manner in which the scope of the execution is defined.

The following subsections details out both the points (Unified Interface definition and Execution Scope definition) and also other principles and concepts that combine to form RESTful Web Services.

A) Concepts

RESTful web services are required to fulfill all the concepts described below:

1) Resource

A resource is an abstract inside the domain that is covered by a service. The service designer has options to choose his desired domain objects, from abstract ones to concrete ones. But at the same time, it is important to keep in mind that any given single resource can be composed of its own separate collection of objects. Also, some category of services may deal with one or more resource type, however, it does not affect the quality of RESTful services in any way.

2) Representation

Web Services has the capability of manipulating the representations of any of the resource, but not so for the resource themselves, as the resources are just an abstractions by themselves. Theoretically, representation is just any useful knowledge about the resource state (Richardson, Ruby 2007). But thinking from a technical point of view, we can see representation as a resource serialization in some given format, such as XML or XHTML or RDF (Resource Description Framework) or JSON (JavaScript Object Notation) and so forth.

3) Uniform Identifier

Any single resource is attached to at least one Uniform Resource Identifier (URI) that simultaneously acts as resource's name and locator. On one hand, if the object does not have a URI, it cannot be considered a resource of RESTful

services. On the other hand, a set of various URI's can identify a single object. The identifiers of the resources need to be a bit expansive and descriptive in a sense that it has to include hierarchy, structure and notation pattern.

4) Unified Interface

According to RESTful web service paradigm, the actions that need to be executed are defined by the HTTP methods. The HTTP protocol has five principal methods, which are, HEAD, GET, POST, DELETE and PUT. All of these methods are applied to the resources. Specifically, any HTTP method when executed is against the provided URI. Despite the simplicity of the characteristic, it is extremely powerful, as it gives a definition of a unified interface to all the services possible. If a client application has the knowledge of the resources that are on offer by a given service, it can automatically know the process of creating, retrieving, deleting and updating these resources.

5) Execution Scope

The RESTful paradigm defines the execution scope by advocating using the HTTP request URI. Hence, not only does the URI contain the service path but also all the other parameters that uniquely identify the affected resource.

B) Principles

RESTful Web Services has to comply as per the principles laid down in addition to the concepts to follow.

1) Addressability

   A Web Service may be called addressable if it can expose the diverse aspects of the data set via resources, each of it with its own unique URI. Assuming that one resource can support different representations, it could be helpful to assign a unique URI to each of the representations. In contrast with this principle, only one URI are typically presented by non-addressable services, which addresses and names the service as a whole, and the objects are not exposed to the client. The latter part is mostly executed by RPC services, which keeps the client from pointing to any resource in particular directly, which is not desirable.

2) Statelessness

   Richardson and Ruby (2007) defines two types of state for RESTful approach, a Resource State and an Application State. A resource state gives the information about the resource whereas an application state is all about the path the user has navigated through the application. This implies that if the server does not store the application state, then the RESTful service is said to be stateless. A client or a user is expected to send all the information required to successfully execute a method.

3) Connectedness

   Resources need to point at each other, which will help in guiding a user to all the other application states. High levels of connectedness empowers discovering of resources.

C) Syntactic Description

A well-defined and established standard for syntactically defining RESTful services are not available yet. However, Hadley (2009) proposed Web Application Description Language (WADL), an XML-based open language that has been in use in recent times.

Chapter 4
REST of the WEB SERVICES

4.1 SOAP

SOAP (Simple Object Access Protocol) based web service is a method that is used to exchange XML messages on internet. A SOAP message is transferred by forming a SOAP-envelope. This envelope elements identifies an XML document. The request and response information is contained in the header element. Procedure calls and messages are defined in the XML document and then sent over using any of the transport protocols. (Tekli et.al 2012).

SOAP based web service has a complex design and hence has been widely criticized. Web Service architecture based on SOAP defines 3 entities: - service registry, service provider and service requester. A Network addressable entity, which executes the requests from a service consumer is a Service Provider. A network-based registry or a directory that has the list of all available services is called a service registry.

4.2 XML-RPC

XML-RPC is a remote procedure calling method that uses HTTP as a transport protocol and XML for the encoding. “XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned.” (xmlrpc.com)

It is a simple and yet very effective means for the request and response processing of the information. It is not a solution to each and every problem though. It encodes and decodes the remote procedure call using XML along with the parameters. Compared to SOAP based web service, XML-RPC has a very simple architecture.

Chapter 5

REST vs SOAP

Once we have looked at what REST and SOAP are, and that we know on what is REST based on, it seems to be more of old age philosophy instead of a new booming technology. It is rather a realization in technology that came way later. Whereas Simple Object Process Protocol seems like a next-gen phase of World Wide Web with a range of new specifications. The REST philosophy emphasizes on the existing protocols and principles that are more than enough to create a robust web service.

Jumping on to comparison between the features of both web services.

The main features of REST web services are:

- Simple to develop: No toolkit is required.
- Lightweight: based on XML without any extra overheads.
- Human Legible results

The main features of SOAP are:

- It has a wide range of development tools.
- It is easier to consume.
- It is rigid, in the sense of type-checking. It has to adhere to a contract.

There are certain points of comparison which will make it easier to differentiate out REST and SOAP based web service:-

A) API Simplicity & Flexibility

The key to writing RESTful Web service is by using a well-known and a widely used interface, i.e. URI. For instance, a simple currency converter, where a user enters currency symbol to return a target currency. To make this script available on a server could be as simple as the following URI:

http://www.CurrencyConverter.com/convert?=us-dollar&value=50&target=Euro

Any client service can call this service easily with HTTP support's GET command. This interface call method has its benefits over SOAP web service. The developer does not need any knowledge to figure out the process for creation and modification of this URI to access resources. On the other hand, for SOAP, there is a need for specific knowledge of its specifications in XML, and also most of them will require a toolkit to create requests and parse responses.

B) Usage of Bandwidth

One more advantage of REST is the interface which can short requests and responses. SOAP needs an XML wrapped around each request and response. Once all the typing and namespaces are defined, a five digit variable in a SOAP message needs at least about 10 times of bytes more than a message in REST.

C) Security

The most interesting and important aspect of REST vs SOAP argument is Security. REST call requests transmit over HTTP or HTTPS. Also, the Firewall can recognize the intention of each XML message by analyzing the HTTP command. For instance, a GET request is considered to be safe as it cannot modify any data, it can only retrieve data.

On the other hand, A SOAP request is a POST method call. So, there is no way to find out whether the request is to retrieve data or to delete a table until it looks into the SOAP enveloper, which in itself is a task that consumes resources and is not a built in feature in most firewalls.

D) Type Handling

SOAP based web service provides relatively much stronger typing as it has a fixed set of data types that are supported. A guarantee of the return value being available directly in any particular platform is therefore achieved. On the other hand, in case of HTTP API, the return value is not guaranteed as such because it has to be deserialized and then type-casted from XML. But this does not mean it needs much effort, even in

complex applications, as traversing down an object is very similar to searching an XML tree. So, there isn't any specific benefits in terms of coding on client-side.

E) Client-side Complexity

Making a request call to an HTTP API is relatively easier than making request calls to SOAP based API. SOAP API needs a stub, a client library and a learning curve. Request calls to REST based API is easy with all programming languages and just involves construction of an HTTP request with parameters appended.

F) Troubleshooting and Testing

It is very easy to troubleshoot and test an API based on HTTP since it can be done with just the browser. No separate troubleshooting tools are needed to generate any requests or responses. This is the main benefit of HTTP based API.

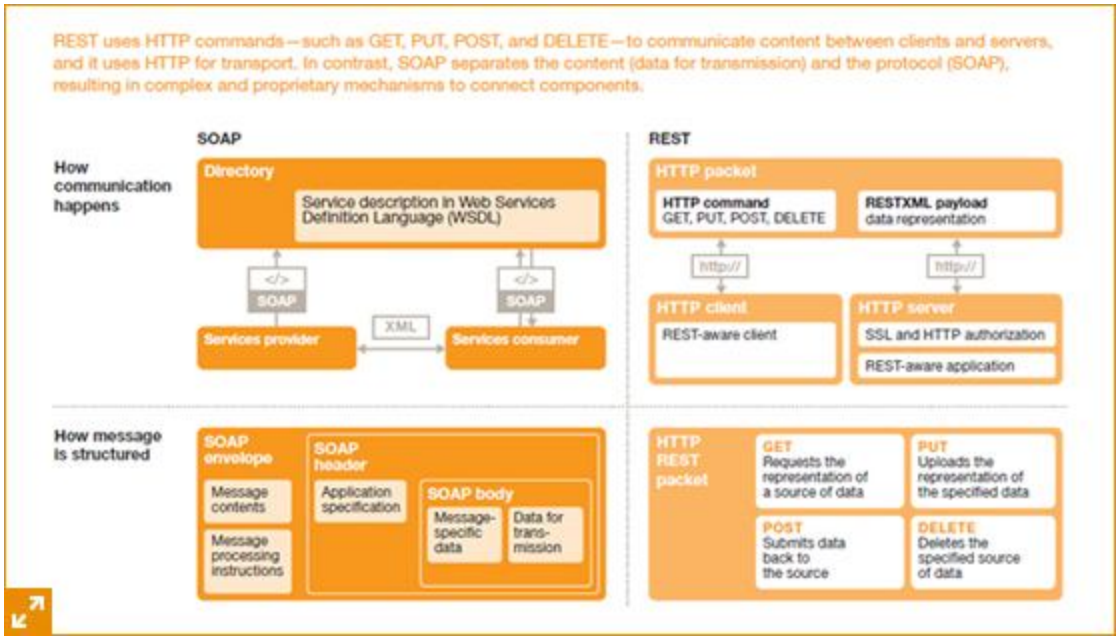


Fig 2. REST vs SOAP (source: pwc.com)

G) Complexity on Server-Side

There are many server side programming languages available today that makes it extremely easier to expose a method in SOAP based web service. SOAP Service library handles the deserialization and serialization. Exposing an object's method in

HTTP API can be a bit more challenging as it requires Serialization of the response to XML.

H) Caching

As RESTful Web Services are able to be consumed using a simple GET request, immediate servers or proxy servers can cache their responses with no extra overhead. On the other hand, SOAP based web services uses POST for its request calls and requires a complex XML document to be created which makes it a bit difficult for immediate proxy servers to cache its responses.

Chapter 6

PERFORMANCE EVALUATION

SOAP based web service has its own various applications in mobile computing environment, multimedia conferencing, etc. The messages in SOAP require large amount of bandwidth, encryption and decryption of XML based messages in SOAP consumes resources. These are a couple of performance overheads which are unacceptable where high performance is concerned. RESTful web services being simple in message transfer is a better alternative to get high performance in such situations. Here, we have used mobile computing and multimedia conferencing as two applications areas to compare the performances of RESTful Web Service and SOAP based Web Service.

In Mobile Devices, Web services are used to connect mobile systems to a distributed computing environment. Consider a mobile application as client service. For benchmarking purposes, we use string and a float data type which will be passed as a parameter in the web service call. The client service is initialized on the emulator. To capture results effectively and accurately, many trials and runs were executed with different values and the results were captured in columns of response time of the server and the message size of the call. It was inference from the tabular data that,

- The message sizes for String and float data types in RESTful Web Services was approximately 10 times smaller than the message size for the same data types in SOAP based web service.
- It was also seen that the processing time and the transmission time required in RESTful Web Service was approximately 5 times less than the time required to process and transmit a message in SOAP web service.

REST Web service		SOAP web service	
Message Size (byte)	**Time (milliseconds)**	**Message Size (byte)**	**Time (milliseconds)**
40	**350**	**350**	**780**
50	**355**	**375**	**830**
65	**360**	**400**	**850**
75	**360**	**425**	**970**

Table 1: Performance Evaluation using String (source: Hamad et.al 2012)

REST Web service		SOAP web service	
Message Size (byte)	**Time (milliseconds)**	**Message Size (byte)**	**Time (milliseconds)**
30	**360**	**355**	**780**
35	**405**	**380**	**780**
36	**408**	**410**	**925**
39	**360**	**435**	**1020**

Table 2: Performance Evaluation using float (source Hamad et.al 2012)

Consider multimedia conferencing, which includes developing RESTful and SOAP based web services for applications such as online games, distance learning, audio/video conferencing, etc. The parlay-x's conferencing model, which is a reference model for Multimedia conferencing is basically based on the entities such as participants, conference type and the media used, where conference type is the unique identifiable context in which a participant can be added and removed, a participant represents the entities who participate in the conference and the media is the stream used to support participants communication. For performance evaluation purposes, we consider different scenarios like conference connection, adding of participants, removing participants, ending a conference. A conferencing gateway is used for the purpose of

implementing the conferencing application API in the respective web service. The request handlers are a part of this gateway used. The data was captured in tabular form, which can be concluded as,

- The time delays from the beginning to the end or from end point to end point was found to be approximately 5 times less in RESTful web services as compared to SOAP based web services.
- The load balancing principles in REST ensures at least 3 times of network load less in RESTful web services than the load in SOAP web service.

Similarly, the sending and receiving of messages in short Messaging service (SMS) uses RESTful Web Services instead of the SOAP based web services. The reason being the handling of requests and responses are much simpler in REST than in SOAP. In SOAP based web services, the messages are written in SOAP format, then enveloped in an HTTP message before transmitting it. Whereas in REST web service, the message only uses HTTP as application layer protocol.

Chapter 7

REASONS WHY REST IS PREFERRED OVER SOAP

In modern times, Enterprise architects are aware of the fact that there is no single middleware solution when it comes to integration of application. But REST based approach is popularizing when it comes to web service development and modularization. REST has enough features to short list for any enterprise application while comparing various tools to develop web service, even though REST may not be an apt solution for each of the situations. A REST based approach can even be effective in the most challenging situations because of its simplicity and ubiquity, where alternatives like SOAP based web service or Java messaging may lose its charm.

7.1 Effective RESTful Techniques

One of the key features of REST is the simplicity it has. But such a simple approach sometimes raises questions such as if this approach will be able to deal with the complex enterprise application challenges that it has to deal with on daily basis. For instance, how about handling real world, complex, transactional applications which involves a number of resources that needs to be in sync for interactions such as an online shopping application. A very effective RESTful technique to solve this issue can be as simple as breaking down the related business logic into all underlying components. The steps involved in an online shopping application includes people being able to browse all products, selecting a product, viewing and editing their online cart, entering payment and shipment information and then checking out by doing the final payment. At any point in the application till the user completes the payment, the user should be able to navigate back to the previous step. From web services API point of view, it means that mapping all the steps of the transaction into resources. In the end or the final step of payment, a complete HTTP request that has all the steps taken by the user or rather, all the states the resources were in, are included in the request.

Also, REST based approach can be used to develop a distributed computing architecture effectively. It becomes very easy to develop a distributed architecture when you have an effective and a proven structure within which an application can be developed. Furthermore, developers prefer the RESTful architecture because of the modularity involved. It allows the development community to have smaller, agile parts of an application that will provide more flexibility in designing solutions that works.

7.2 Critique

Looking at the small and simple size of REST messages along with the worldwide adopted HTTP, it makes REST a very attractive choice for API's. On the other hand, SOAP has something more than just the request response messages. It has a set of standard specifications called the WSDL. Each and every thing used in a SOAP request and a response message has to be clearly defined and specified in the WSDL document. And each and every change in WSDL document also means changes on the client side. SOAP can be useful in situations where the size of the message and does not matter or where the application needs to be controlled from end-to-end. For everything else, REST is always a simpler and flexible alternative.

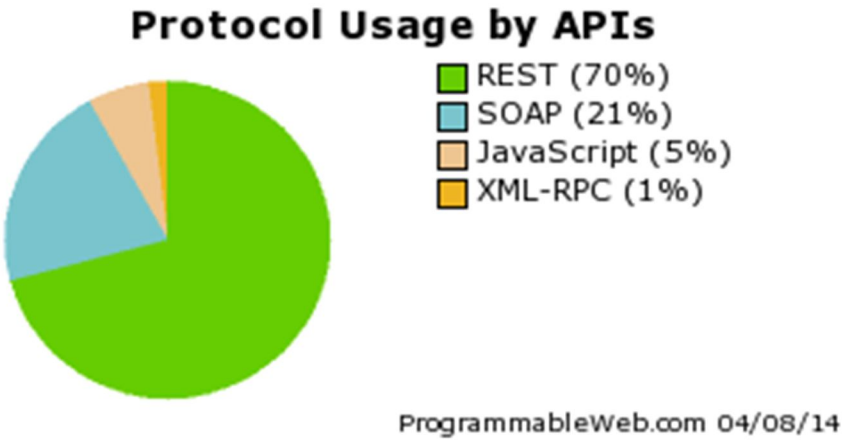


Fig 3. Pie-Chart of Web services being used (Source: programmableweb.com, 2014)

The figure above illustrates the usage of RESTful based web services has increased to more than thrice the usage of SOAP based web services.

Chapter 8

Web Service MESSAGES - JSON Parsed

8.1 REST REQUESTS AND RESPONSES

1. Get All Messages
   Request:

   GET /messenger/webapi/messages HTTP/1.1
   Host: localhost:8080
   Content-Type: application/json
   Cache-Control: no-cache
   Postman-Token: 6788c2d9-2bb9-9519-f2c5-57aa69d61612

   Response:

```
[
    {
        "author": "Dhruv",
        "created": "2015-08-04T00:00:00-04:00",
        "id": 98,
        "links": [],
        "message": "Hello DB"
    },
    {
        "author": "Dhruv",
        "created": "2015-07-22T00:00:00-04:00",
        "id": 99,
        "links": [],
        "message": "This is a TEST message"
    }
]
```


2. Get a single message
   Request:
   GET /messenger/webapi/messages/98 HTTP/1.1
   Host: localhost:8080
   Content-Type: application/json
   Cache-Control: no-cache
   Postman-Token: 74c25e4b-2659-ea2f-84be-3e1175d363bd

   Response:

```
{
  "author": "Dhruv",
  "created": "2015-08-04T00:00:00-04:00",
```

```
 "id": 98,
 "links": [
  {
   "link": "http://localhost:8080/messenger/webapi/messages/98",
   "rel": "self"
  },
  {
   "link": "http://localhost:8080/messenger/webapi/profiles/Dhruv",
   "rel": "profile"
  },
  {
   "link": "http://localhost:8080/messenger/webapi/messages/98/comments/",
   "rel": "comment"
  }
 ],
 "message": "Hello DB"
}
```


3. Post a message

Request:

```
POST /messenger/webapi/messages HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Cache-Control: no-cache
Postman-Token: 2cabfb0c-3371-459f-7279-59258b64557e

{
   "author": "DRD",
   "created": "2015-08-04T00:00:00-04:00",
   "id": 100,
   "links": [],
   "message": "Testing Post messages"
 }
```

Response:


```
{
 "author": "DRD",
 "created": "2015-08-04T00:00:00-04:00",
 "id": 100,
 "links": [],
 "message": "Testing Post messages"
}
```


4. Update a message

Request:

```
PUT /messenger/webapi/messages/100 HTTP/1.1
Host: localhost:8080
```

Content-Type: application/json
Cache-Control: no-cache
Postman-Token: 304f5f77-9c6c-d8ab-0682-93782f5a1d3e

{
"author": "DRD",
"created": "2015-08-04T00:00:00-04:00",
"id": 100,
"links": [],
"message": "Updating messages"
}

Response:
{
"author": "DRD",
"created": "2015-08-04T00:00:00-04:00",
"id": 100,
"links": [],
"message": "Updating messages"
}

5. Delete a message
Request:
DELETE /messenger/webapi/messages/100 HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Cache-Control: no-cache
Postman-Token: e8a39e67-7a42-bff0-3ac2-802a9e36e276

{
"author": "DRD",
"created": "2015-08-04T00:00:00-04:00",
"id": 100,
"links": [],
"message": "Deleting messages"
}

Response:
Status: 204 No Content

6. Get all the comments for a message
Request:
GET /messenger/webapi/messages/98/comments HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Cache-Control: no-cache
Postman-Token: 21b8e0de-5caa-8ae9-7f9c-a33f822d6a3f

Response:


```
[
  {
    "author": "Dhruv",
    "created": "2015-08-18T00:00:00-04:00",
    "id": 1,
    "message": "This is a Comment"
  },
  {
    "author": "Dhruv",
    "created": "2015-08-18T00:00:00-04:00",
    "id": 2,
    "message": "Test"
  }
]
```


7. Get a specific comment for a specific message

Request:

```
GET /messenger/webapi/messages/98/comments/1 HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Cache-Control: no-cache
Postman-Token: b5a14ff2-b7f4-d8d6-e28f-af195112f1e8
```

Response:


```
{
  "author": "Dhruv",
  "created": "2015-08-18T00:00:00-04:00",
  "id": 1,
  "message": "This is a Comment"
}
```


8. Post a comment

Request:

```
POST /messenger/webapi/messages/98/comments HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Cache-Control: no-cache
Postman-Token: 198844e1-25b4-7e4b-d099-9a7c82ab11fa
```


```
{
  "author": "DRD",
  "created": "2015-08-18T00:00:00-04:00",
  "id": 3,
  "message": "Comments testing"
}
```

Response:

```
{
  "author": "DRD",
  "created": "2015-08-18T00:00:00-04:00",
  "id": 3,
  "message": "Comments testing"
}
```


9. Get all profiles

Request:
```
GET /messenger/webapi/profiles HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Cache-Control: no-cache
Postman-Token: bfe80e13-72f6-ad93-e6ca-968c37a676bf
```

Response:

```
[
  {
    "created": "2015-09-03T00:00:00-04:00",
    "firstName": "Dhruv",
    "id": 1,
    "lastName": "Desai",
    "profileName": "admin"
  }
]
```

Chapter 9

Application Code

9.1 Model Package

9.1.1 Message Model

```
package org.dhruv.rest.messenger.model;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
@XmlRootElement
public class Message {
	private long id;
	private String message;
	private Date created;
	private String author;
	private Map<Long, Comment> comments = new HashMap<>();
	private List<Link> links = new ArrayList<>();
	public Message(long id, String message, String author) {
		super();
		this.id = id;
		this.message = message;
		this.created = new Date();
		this.author = author;
	}
	public long getId() {
```

```
        return id;
}
public void setId(long id) {
        this.id = id;
}
public String getMessage() {
        return message;
}
public void setMessage(String message) {
        this.message = message;
}
public Date getCreated() {
        return created;
}
public void setCreated(Date created) {
        this.created = created;
}
public String getAuthor() {
        return author;
}
public void setAuthor(String author) {
        this.author = author;
}
@XmlTransient
public Map<Long, Comment> getComments() {
        return comments;
}
public void setComments(Map<Long, Comment> comments) {
        this.comments = comments;
}
```

```
	public List<Link> getLinks() {
		return links;
	}
	public void setLinks(List<Link> links) {
		this.links = links;
	}
	public void addLink(String url, String rel){
		Link link = new Link();
		link.setLink(url);
		link.setRel(rel);
		links.add(link);
	}
}
```

9.1.2 Profile Model

```
package org.dhruv.rest.messenger.model;
import java.util.Date;
import javax.xml.bind.annotation.XmlRootElement;
@XmlRootElement
public class Profile {
	private long id;
	private String profileName;
	private String firstName;
	private String lastName;
	private Date created;
	public Profile(long id, String profileName, String firstName,
				String lastName) {
		super();
		this.id = id;
		this.profileName = profileName;
		this.firstName = firstName;
		this.lastName = lastName;
		this.created = new Date();
	}
	public long getId() {
		return id;
	}
	public void setId(long id) {
		this.id = id;
```

```
    }
    public String getProfileName() {
        return profileName;
    }
    public void setProfileName(String profileName) {
        this.profileName = profileName;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public Date getCreated() {
        return created;
    }
    public void setCreated(Date created) {
        this.created = created;
    }
}
```

9.1.3 Comment Model

```
package org.dhruv.rest.messenger.model;
import java.util.Date;
public class Comment {
    private long id;
  private String message;
  private Date created;
  private String author;
  public Comment(long id, String message, String author) {
    this.id = id;
    this.message = message;
    this.author = author;
    this.created = new Date();
  }
    public long getId() {
        return id;
    }
    public void setId(long id) {
```

```
		this.id = id;
	}
	public String getMessage() {
		return message;
	}
	public void setMessage(String message) {
		this.message = message;
	}
	public Date getCreated() {
		return created;
	}
	public void setCreated(Date created) {
		this.created = created;
	}
	public String getAuthor() {
		return author;
	}
	public void setAuthor(String author) {
		this.author = author;
	}
}
```

9.1.4 Link Model

```
package org.dhruv.rest.messenger.model;
public class Link {
	private String link;
	private String rel;
	public String getLink() {
		return link;
	}
	public void setLink(String link) {
		this.link = link;
	}
	public String getRel() {
		return rel;
	}
	public void setRel(String rel) {
		this.rel = rel;
	}

}
```

9.2 Resource Package

9.2.1 Message Resource

```
package org.dhruv.rest.messenger.resources;
import java.net.URI;
import java.util.List;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;
import javax.ws.rs.core.UriInfo;
import org.dhruv.rest.messenger.model.Message;
import org.dhruv.rest.messenger.service.MessageService;
@Path("/messages")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public class MessageResource {
        MessageService messageservice = new MessageService();
        @GET
        public List<Message> getMessages(@QueryParam("year") int year,
```

```
                                                    @QueryParam("start") int
start,
                                                    @QueryParam("size") int
size){
            if(year>0){
                  return messageservice.getAllMessagesForYear(year);
            }
            if(start>=0 && size>0){
                  return messageservice.getAllMessagesPaginated(start, size);
            }
            return messageservice.getAllMessages();
      }
      @GET
      @Path("/{messageId}")
      public Message getMessage(@PathParam("messageId") Long messageId, @Context
UriInfo uriInfo){
            Message message = messageservice.getMessage(messageId);
            message.addLink(getUriForSelf(uriInfo, message), "self");
            message.addLink(getUriForProfile(uriInfo, message), "profile");
            message.addLink(getUriForComment(uriInfo, message), "comment");
            return message;
      }
      private String getUriForComment(UriInfo uriInfo, Message message) {
            return uriInfo.getBaseUriBuilder()
                        .path(MessageResource.class)
                        .path(MessageResource.class, "getCommentResource")
                        .path(CommentResource.class)
                        .resolveTemplate("messageId", message.getId())
                        .build()
                        .toString();
      }
```

```
	private String getUriForProfile(UriInfo uriInfo, Message message) {
		return uriInfo.getBaseUriBuilder()
					.path(ProfileResource.class)
					.path(message.getAuthor())
					.build()
					.toString();
	}
	public String getUriForSelf(UriInfo uriInfo, Message message) {
		return uriInfo.getBaseUriBuilder()
					.path(MessageResource.class)
					.path(Long.toString(message.getId()))
					.build()
					.toString();
	}
	@PUT
	@Path("/{messageId}")
	public Message updateMessage(@PathParam("messageId") Long messageId, Message
message){
		message.setId(messageId);
		return messageservice.updateMessage(message);
	}
	@POST
	public Response addMessage(Message message, @Context UriInfo uriInfo){
		Message newMessage = messageservice.addMessage(message);
		String newId = String.valueOf(newMessage.getId());
		URI uri = uriInfo.getAbsolutePathBuilder().path(newId).build();
		return Response.created(uri)
				.entity(newMessage)
				.build();
	}
```

```
	@DELETE
	@Path("/{messageId}")
	public void deleteMessage(@PathParam("messageId") Long messageId){
		messageservice.removeMessage(messageId);
	}
	@Path("/{messageId}/comments")
	public CommentResource getCommentResource(){
		return new CommentResource();
	}
}
```

9.2.2 Comment Resource

```
package org.dhruv.rest.messenger.resources;
import java.util.List;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import org.dhruv.rest.messenger.model.Comment;
import org.dhruv.rest.messenger.service.CommentService;
@Path("/")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class CommentResource {
```

```
	private CommentService commentservice = new CommentService();

	@GET

	public List<Comment> getAllComments(@PathParam("messageId") int messageId){

		return commentservice.getAllComments(messageId);

	}

	@GET

	@Path("/{commentId}")

	public Comment getComment(@PathParam("messageId") int messageId,
@PathParam("commentId") int commentId){

		return commentservice.getComment(messageId, commentId);

	}

	@POST

	public Comment addComment(@PathParam("messageId") int messageId, Comment
comment){

		//System.out.println("CommentResource method");

		return commentservice.addComment(messageId, comment);

	}

	@PUT

	@Path("/{commentId}")

	public Comment updateComment(@PathParam("messageId") int messageId,
@PathParam("commentId") int commentId, Comment comment){

		comment.setId(commentId);

		return commentservice.updateComment(messageId, comment);

	}

	@DELETE

	@Path("/{commentId}")

	public void deleteComment(@PathParam("messageId") int messageId,
@PathParam("commentId") int commentId){

		commentservice.removeComment(messageId, commentId);

	}

}
```

9.2.3 Profile Resource

```
package org.dhruv.rest.messenger.resources;
import java.util.List;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import org.dhruv.rest.messenger.model.Profile;
import org.dhruv.rest.messenger.service.ProfileService;
@Path("/profiles")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public class ProfileResource {
	ProfileService profileservice = new ProfileService();
	@GET
	public List<Profile> getProfiles(){
		return profileservice.getAllProfiles();
	}
	@GET
	@Path("/{profileName}")
	public Profile getProfile(@PathParam("profileName") String profileName){
		return profileservice.getProfile(profileName);

	}
	@POST
```

```
	public Profile addProfile(Profile profile){
		return profileservice.addProfile(profile);
	}
	@PUT
	@Path("/{profileName}")
	public Profile updateProfile(@PathParam("profileName") String profileName, Profile
profile){
		profile.setProfileName(profileName);
		return profileservice.updateProfile(profile);
	}
	@DELETE
	@Path("/{profileName}")
	public void deleteProfile(@PathParam("profileName") String profileName){
	  profileservice.removeProfile(profileName);
	}
}
```

9.3 Service Package

9.3.1 MessageService

```
package org.dhruv.rest.messenger.service;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import java.util.Map;
import org.dhruv.rest.messenger.database.DatabaseClass;
import org.dhruv.rest.messenger.database.MessageUpdateObject;
import org.dhruv.rest.messenger.exception.DataNotFoundException;
import org.dhruv.rest.messenger.model.Message;
public class MessageService {
```

```
MessageUpdateObject uo = new MessageUpdateObject();
public List<Message> getAllMessages(){
        //Message s1 = new Message(1L, "Hello World", "Dhruv");
        //Message s2 = new Message(2L, "Hello Jersey", "Dhruv");
        //List<Message> list = new ArrayList<>();
        //list.add(s1);
        //list.add(s2);
  //return new ArrayList<Message>(messages.values());
        return uo.getAllMessages();
}
Public List<Message> getAllMessagesForYear(int year){
        List<Message> messagesForYear = new ArrayList<>();
        List<Message> messages = new ArrayList<>();
        messages = uo.getAllMessages();
        Calendar cal = Calendar.getInstance();
        for(Message message : messages){
                cal.setTime(message.getCreated());
                if(cal.get(Calendar.YEAR) == year)
                        messagesForYear.add(message);
        }
        return messagesForYear;
}
public List<Message> getAllMessagesPaginated(int start, int size){
        List<Message> list = new ArrayList<Message>(uo.getAllMessages());
        if(start + size > list.size())
                return new ArrayList<Message>();
        return list.subList(start, start+size);
}
public Message getMessage(long id){
        //List<Message> msg = new ArrayList<>();
```

```
		//msg = uo.getAllMessages();
		Message message = uo.getMessage(id);
		if(message == null){
			throw new DataNotFoundException("Message with id "+id+" not found");
		}
		return message;
	}
	public Message addMessage(Message message){
		//message.setId(messages.size() + 1);
		//messages.put(message.getId(), message);
		uo.insertMessage(message);
		return message;
	}
	public Message updateMessage(Message message){
		if (message.getId() <= 0){
			return null;
		}
		//messages.put(message.getId(), message);
		uo.updateMessage(message);
		return message;
	}
	public void removeMessage(long id){
		//return messages.remove(id);
		uo.deleteMessage(id);
	}
}
```

9.3.2 CommentService

```
package org.dhruv.rest.messenger.service;
import java.util.ArrayList;
import java.util.List;
```

```
import java.util.Map;
import org.dhruv.rest.messenger.database.CommentUpdateObject;
import org.dhruv.rest.messenger.database.DatabaseClass;
import org.dhruv.rest.messenger.model.Comment;
import org.dhruv.rest.messenger.model.Message;
public class CommentService {
	private Map<Long, Message> messages = DatabaseClass.getMessages();
	CommentUpdateObject uo = new CommentUpdateObject();
	public List<Comment> getAllComments(long messageId) {
		//Map<Long, Comment> comments = messages.get(messageId).getComments();
		//return new ArrayList<Comment>(comments.values());
		return uo.getAllComments(messageId);
	}
	public Comment getComment(long messageId, long commentId) {
		return uo.getComment(messageId, commentId);
	}
	public Comment addComment(long messageId, Comment comment) {
		uo.insertComment(messageId, comment);
		return comment;
	}
	public Comment updateComment(long messageId, Comment comment) {
		Map<Long, Comment> comments = messages.get(messageId).getComments();
		if (comment.getId() <= 0) {
			return null;
		}
		comments.put(comment.getId(), comment);
		return comment;
	}
	public void removeComment(long messageId, long commentId) {
		uo.deleteComment(commentId);
```

```
	}
}
```

9.3.3 ProfileService

```
package org.dhruv.rest.messenger.service;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import org.dhruv.rest.messenger.database.DatabaseClass;
import org.dhruv.rest.messenger.database.ProfileUpdateObject;
import org.dhruv.rest.messenger.model.Message;
import org.dhruv.rest.messenger.model.Profile;
public class ProfileService {
	ProfileUpdateObject uo = new ProfileUpdateObject();
	public ProfileService(){
		//profiles.put("Dhruv", new Profile(1L, "Dhruv", "Dhruv", "Desai"));
	}
	public List<Profile> getAllProfiles(){
		//Message s1 = new Message(1L, "Hello World", "Dhruv");
		//Message s2 = new Message(2L, "Hello Jersey", "Dhruv");
		//List<Message> list = new ArrayList<>();
		//list.add(s1);
		//list.add(s2);
		//return new ArrayList<Profile>(profiles.values());
		return uo.getAllProfiles();
	}
	public Profile getProfile(String profileName){
		//return profiles.get(profileName);
		return uo.getProfile(profileName);
	}
	public Profile addProfile(Profile profile){
```

```
                //profile.setId(profiles.size() + 1);
                //profiles.put(profile.getProfileName(), profile);
                uo.insertProfile(profile);
                return profile;
        }
        public Profile updateProfile(Profile profile){
                if (profile.getProfileName().isEmpty()){
                        return null;
                }
                //profiles.put(profile.getProfileName(), profile);
                uo.updateProfile(profile);
                return profile;
        }
        public void removeProfile(String profileName){
                //return profiles.remove(profileName);
                uo.deleteProfile(profileName);
        }
}
```

9.4 Database Package

9.4.1 MessageUpdateObject

```
package org.dhruv.rest.messenger.database;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
```

```
import java.util.List;

import org.dhruv.rest.messenger.model.Message;

public class MessageUpdateObject {

	static final String DB_URL = "jdbc:mysql://localhost/messenger";
   static final String USER = "root";
   static final String PASS = "root";
   Connection conn = null;
   Statement stmt = null;
   PreparedStatement pstmt= null;

   public List<Message> getAllMessages(){
		List<Message> list = new ArrayList<>();
		try{
			Class.forName("com.mysql.jdbc.Driver");
			conn = DriverManager.getConnection(DB_URL, USER, USER);
			stmt = conn.createStatement();
			String sql = "SELECT MessageId, Message, Author, Created_Date from
message"; //where MessageId = ?";
			ResultSet rs = stmt.executeQuery(sql);
			while(rs.next()){
				Message newmsg = new Message();
				newmsg.setId(rs.getInt("MessageId"));
				newmsg.setMessage(rs.getString("Message"));
				newmsg.setAuthor(rs.getString("Author"));
				newmsg.setCreated(rs.getDate("Created_Date"));
				list.add(newmsg);
			}
```

```
            }catch(Exception e){
                e.printStackTrace();
            }finally{
                try{
                    if(stmt!=null)
                        stmt.close();
                    if(conn!=null)
                        conn.close();
                }catch(SQLException se2){
                }
            }
            return list;
        }

    public Message getMessage(long id){
            //List<Message> list = new ArrayList<>();
        Message newmsg = new Message();
            try{
                Class.forName("com.mysql.jdbc.Driver");
                conn = DriverManager.getConnection(DB_URL, USER, USER);
                String sql = "SELECT MessageId, Message, Author, Created_Date from
message where MessageId = ?";
                pstmt = conn.prepareStatement(sql);
                pstmt.setLong(1, id);
                ResultSet rs = pstmt.executeQuery();
                while(rs.next()){
                    newmsg.setId(rs.getInt("MessageId"));
                    newmsg.setMessage(rs.getString("Message"));
                    newmsg.setAuthor(rs.getString("Author"));
                    newmsg.setCreated(rs.getDate("Created_Date"));
```

```
                        //list.add(newmsg);
                }
            }catch(Exception e){
                e.printStackTrace();
            }finally{
                try{
                    if(pstmt!=null)
                        pstmt.close();
                    if(conn!=null)
                        conn.close();
                }catch(SQLException se2){
                }
            }
            return newmsg;
        }
    public void insertMessage(Message message){
        try{
            Class.forName("com.mysql.jdbc.Driver");
                conn = DriverManager.getConnection(DB_URL, USER, USER);
                String sql = "Insert into Message values (?,?,?,?)";
                pstmt = conn.prepareStatement(sql);
                pstmt.setLong(1, message.getId());
                pstmt.setString(2, message.getMessage());
                pstmt.setString(3, message.getAuthor());
                //pstmt.setDate(4, new java.sql.Date(message.getCreated().getTime()));
                pstmt.setDate(4, new java.sql.Date(System.currentTimeMillis()));
                pstmt.executeUpdate();
                //conn.commit();
        }catch(Exception e){
            e.printStackTrace();
```

```
        }finally{
                        try{
                                if(pstmt!=null)
                                        pstmt.close();
                                if(conn!=null)
                                        conn.close();
                        }catch(SQLException se2){
                                se2.printStackTrace();
                        }
                }
   }
public void updateMessage(Message message){
        try{
                Class.forName("com.mysql.jdbc.Driver");
                        conn = DriverManager.getConnection(DB_URL, USER, USER);
                        String sql = "update message set Message = ?, Author =? WHERE
MessageId = ?";
                        pstmt = conn.prepareStatement(sql);
                        pstmt.setString(1, message.getMessage());
                        pstmt.setString(2, message.getAuthor());
                        pstmt.setLong(3, message.getId());
                        pstmt.executeUpdate();
        }catch(Exception e){
                e.printStackTrace();
        }finally{
                        try{
                                if(pstmt!=null)
                                        pstmt.close();
                                if(conn!=null)
                                        conn.close();
```

```
                }catch(SQLException se2){
                        se2.printStackTrace();
                }
        }
    }
    public void deleteMessage(long id){
        try{
                Class.forName("com.mysql.jdbc.Driver");
                        conn = DriverManager.getConnection(DB_URL, USER, USER);
                        String sql = "DELETE FROM MESSAGE WHERE MessageId = ?";
                        pstmt = conn.prepareStatement(sql);
                        pstmt.setLong(1, id);
                        pstmt.executeUpdate();
        }catch(Exception e){
                e.printStackTrace();
        }finally{
                        try{
                                if(pstmt!=null)
                                        pstmt.close();
                                if(conn!=null)
                                        conn.close();
                        }catch(SQLException se2){
                                se2.printStackTrace();
                        }
                }
    }
}
```

9.4.2 CommentUpdateObject

```
package org.dhruv.rest.messenger.database;
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

import org.dhruv.rest.messenger.model.Comment;
import org.dhruv.rest.messenger.model.Message;

public class CommentUpdateObject {

        static final String DB_URL = "jdbc:mysql://localhost/messenger";
   static final String USER = "root";
   static final String PASS = "root";
   Connection conn = null;
   Statement stmt = null;
   PreparedStatement pstmt= null;

   public List<Comment> getAllComments(long messageId){
                List<Comment> list = new ArrayList<>();
                try{
                        Class.forName("com.mysql.jdbc.Driver");
                        conn = DriverManager.getConnection(DB_URL, USER, USER);
                        //stmt = conn.createStatement();
                        String sql = "SELECT Comment_Id, Comment, Author, Created_Date
from comment where Message_Id = ?";
                        pstmt = conn.prepareStatement(sql);
```

```
                pstmt.setLong(1, messageId);
                ResultSet rs = pstmt.executeQuery();
                while(rs.next()){
                        Comment newcmt = new Comment();
                        newcmt.setId(rs.getInt("Comment_Id"));
                        newcmt.setMessage(rs.getString("Comment"));
                        newcmt.setAuthor(rs.getString("Author"));
                        newcmt.setCreated(rs.getDate("Created_Date"));
                        list.add(newcmt);
                }
        }catch(Exception e){
                e.printStackTrace();
        }finally{
                try{
                        if(pstmt!=null)
                                pstmt.close();
                        if(conn!=null)
                                conn.close();
                }catch(SQLException se2){
                }
        }
        return list;
    }

public Comment getComment(long messageid, long commentid){
        //List<Message> list = new ArrayList<>();
    Comment newcmt = new Comment();
        try{
                Class.forName("com.mysql.jdbc.Driver");
                conn = DriverManager.getConnection(DB_URL, USER, USER);
```

```
                String sql = "SELECT Comment_Id, Comment, Author, Created_Date
from comment where Message_Id = ? and Comment_Id = ?";
                pstmt = conn.prepareStatement(sql);
                pstmt.setLong(1, messageid);
                pstmt.setLong(2, commentid);
                ResultSet rs = pstmt.executeQuery();
                while(rs.next()){
                        newcmt.setId(rs.getInt("Comment_Id"));
                        newcmt.setMessage(rs.getString("Comment"));
                        newcmt.setAuthor(rs.getString("Author"));
                        newcmt.setCreated(rs.getDate("Created_Date"));
                        //list.add(newmsg);
                }
        }catch(Exception e){
                e.printStackTrace();
        }finally{
                try{
                        if(pstmt!=null)
                                pstmt.close();
                        if(conn!=null)
                                conn.close();
                }catch(SQLException se2){
                }
        }
        return newcmt;
    }

  public void insertComment(long messageId, Comment comment){
    try{
        Class.forName("com.mysql.jdbc.Driver");
```

```
                    conn = DriverManager.getConnection(DB_URL, USER, USER);
                    String sql = "Insert into Comment values (?,?,?,?,?)";
                    pstmt = conn.prepareStatement(sql);
                    pstmt.setLong(1, comment.getId());
                    pstmt.setLong(2, messageId);
                    pstmt.setString(3, comment.getMessage());
                    pstmt.setString(4, comment.getAuthor());
                    //pstmt.setDate(4, new java.sql.Date(message.getCreated().getTime()));
                    pstmt.setDate(5, new java.sql.Date(System.currentTimeMillis()));
                    pstmt.executeUpdate();
                    //conn.commit();
        }catch(Exception e){
                e.printStackTrace();
        }finally{
                    try{
                            if(pstmt!=null)
                                    pstmt.close();
                            if(conn!=null)
                                    conn.close();
                    }catch(SQLException se2){
                            se2.printStackTrace();
                    }
                }
}

public void updateComment(Comment comment){
        try{
                Class.forName("com.mysql.jdbc.Driver");
                    conn = DriverManager.getConnection(DB_URL, USER, USER);
```

```
                String sql = "update message set Message = ?, Author =? WHERE
MessageId = ?";
                pstmt = conn.prepareStatement(sql);
                pstmt.setString(1, comment.getMessage());
                pstmt.setString(2, comment.getAuthor());
                pstmt.setLong(3, comment.getId());
                pstmt.executeUpdate();
        }catch(Exception e){
                e.printStackTrace();
        }finally{
                try{
                        if(pstmt!=null)
                                pstmt.close();
                        if(conn!=null)
                                conn.close();
                }catch(SQLException se2){
                        se2.printStackTrace();
                }
        }
    }
    public void deleteComment(long id){
        try{
                Class.forName("com.mysql.jdbc.Driver");
                        conn = DriverManager.getConnection(DB_URL, USER, USER);
                        String sql = "DELETE FROM COMMENT WHERE Comment_Id = ?";
                        pstmt = conn.prepareStatement(sql);
                        pstmt.setLong(1, id);
                        pstmt.executeUpdate();
        }catch(Exception e){
                e.printStackTrace();
```

```
        }finally{
                        try{
                                if(pstmt!=null)
                                        pstmt.close();
                                if(conn!=null)
                                        conn.close();
                        }catch(SQLException se2){
                                se2.printStackTrace();
                        }
                }
   }
}
```

9.4.3 ProfileUpdateObject

```
package org.dhruv.rest.messenger.database;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;


import org.dhruv.rest.messenger.model.Profile;

public class ProfileUpdateObject {
        static final String DB_URL = "jdbc:mysql://localhost/messenger";
   static final String USER = "root";
```

```
    static final String PASS = "root";
    Connection conn = null;
    Statement stmt = null;
    PreparedStatement pstmt= null;


    public List<Profile> getAllProfiles(){
            List<Profile> list = new ArrayList<>();
            try{
                    Class.forName("com.mysql.jdbc.Driver");
                    conn = DriverManager.getConnection(DB_URL, USER, USER);
                    //stmt = conn.createStatement();
                    String sql = "SELECT Profile_Id, Profile_Name, First_Name, Last_Name,
Created_Date from PROFILE";
                    pstmt = conn.prepareStatement(sql);
                    //pstmt.setLong(1, messageId);
                    ResultSet rs = pstmt.executeQuery();
                    while(rs.next()){
                            Profile profile = new Profile();
                            profile.setId(rs.getInt("Profile_Id"));
                            profile.setProfileName(rs.getString("Profile_Name"));
                            profile.setFirstName(rs.getString("First_Name"));
                            profile.setLastName(rs.getString("Last_Name"));
                            profile.setCreated(rs.getDate("Created_Date"));
                            list.add(profile);
                    }
            }catch(Exception e){
                    e.printStackTrace();
            }finally{
                    try{
                            if(pstmt!=null)
```

```
                                pstmt.close();
                        if(conn!=null)
                                conn.close();
                    }catch(SQLException se2){
                    }
                }
                return list;
        }
    public Profile getProfile(String profileName){
                //List<Message> list = new ArrayList<>();
        Profile profile = new Profile();
                try{
                        Class.forName("com.mysql.jdbc.Driver");
                        conn = DriverManager.getConnection(DB_URL, USER, USER);
                        String sql = "SELECT Profile_Id, Profile_Name, First_Name, Last_Name
,Created_Date from Profile where Profile_Name = ? ";
                        pstmt = conn.prepareStatement(sql);
                        pstmt.setString(1, profileName);
                        ResultSet rs = pstmt.executeQuery();
                        while(rs.next()){
                                profile.setId(rs.getInt("Profile_Id"));
                                profile.setProfileName(rs.getString("Profile_Name"));
                                profile.setFirstName(rs.getString("First_Name"));
                                profile.setLastName(rs.getString("Last_Name"));
                                profile.setCreated(rs.getDate("Created_Date"));
                                //list.add(newmsg);
                        }
                }catch(Exception e){
                        e.printStackTrace();
                }finally{
```

```
                try{
                        if(pstmt!=null)
                                pstmt.close();
                        if(conn!=null)
                                conn.close();
                }catch(SQLException se2){
                }
        }
        return profile;
    }

public void insertProfile(Profile profile){
    try{
        Class.forName("com.mysql.jdbc.Driver");
                conn = DriverManager.getConnection(DB_URL, USER, USER);
                String sql = "Insert into Profile values (?,?,?,?,?)";
                pstmt = conn.prepareStatement(sql);
                pstmt.setLong(1, profile.getId());
                pstmt.setString(2, profile.getProfileName());
                pstmt.setString(3, profile.getFirstName());
                pstmt.setString(4, profile.getLastName());
                //pstmt.setDate(4, new java.sql.Date(message.getCreated().getTime()));
                pstmt.setDate(5, new java.sql.Date(System.currentTimeMillis()));
                pstmt.executeUpdate();
                //conn.commit();
    }catch(Exception e){
        e.printStackTrace();
    }finally{
                try{
                        if(pstmt!=null)
```

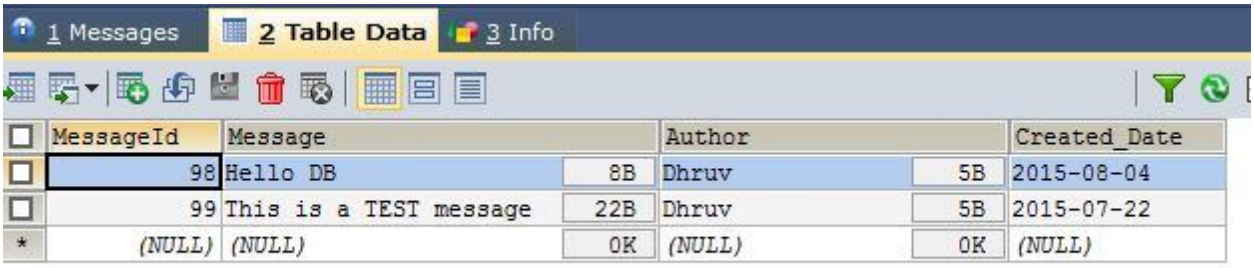
```
                                pstmt.close();
                        if(conn!=null)
                                conn.close();
                    }catch(SQLException se2){
                        se2.printStackTrace();
                    }
                }
    }

    public void updateProfile(Profile profile){
        try{
            Class.forName("com.mysql.jdbc.Driver");
                    conn = DriverManager.getConnection(DB_URL, USER, USER);
                    String sql = "update profile set First_Name = ?, Last_Name =? WHERE
Profile_Name = ?";
                    pstmt = conn.prepareStatement(sql);
                    pstmt.setString(1, profile.getFirstName());
                    pstmt.setString(2, profile.getLastName());
                    pstmt.setString(3, profile.getProfileName());
                    pstmt.executeUpdate();
        }catch(Exception e){
            e.printStackTrace();
        }finally{
                    try{
                        if(pstmt!=null)
                                pstmt.close();
                        if(conn!=null)
                                conn.close();
                    }catch(SQLException se2){
                        se2.printStackTrace();
```

```
                }
            }
    }
    public void deleteProfile(String profileName){
        try{
            Class.forName("com.mysql.jdbc.Driver");
                conn = DriverManager.getConnection(DB_URL, USER, USER);
                String sql = "DELETE FROM PROFILE WHERE Profile_Name = ?";
                pstmt = conn.prepareStatement(sql);
                pstmt.setString(1, profileName);
                pstmt.executeUpdate();
        }catch(Exception e){
            e.printStackTrace();
        }finally{
                try{
                    if(pstmt!=null)
                        pstmt.close();
                    if(conn!=null)
                        conn.close();
                }catch(SQLException se2){
                    se2.printStackTrace();
                }
            }
    }
```

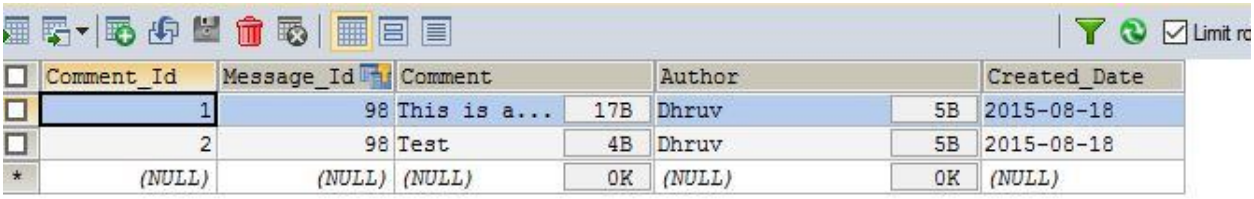
Chapter 10
DATABASE

1. Message Table :



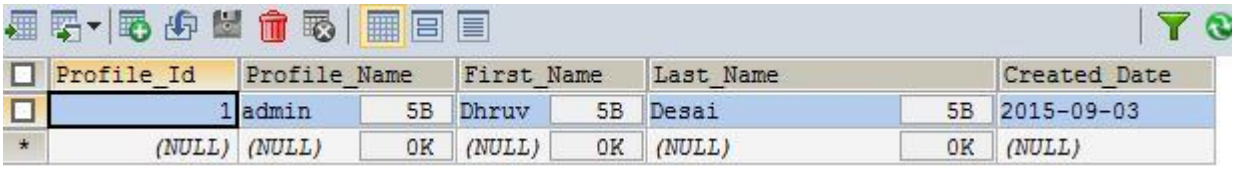
MessageId	Message		Author		Created_Date
98	Hello DB	8B	Dhruv	5B	2015-08-04
99	This is a TEST message	22B	Dhruv	5B	2015-07-22
(NULL)	*(NULL)*	0K	*(NULL)*	0K	*(NULL)*

2. Comment Table:



Comment_Id	Message_Id	Comment		Author		Created_Date
1	98	This is a...	17B	Dhruv	5B	2015-08-18
2	98	Test	4B	Dhruv	5B	2015-08-18
(NULL)	*(NULL)*	*(NULL)*	0K	*(NULL)*	0K	*(NULL)*

3. Profile Table:



Profile_Id	Profile_Name		First_Name		Last_Name		Created_Date
1	admin	5B	Dhruv	5B	Desai	5B	2015-09-03
(NULL)	*(NULL)*	0K	*(NULL)*	0K	*(NULL)*	0K	*(NULL)*

CONCLUSION

As we are aware of the fact that web services are widely used all over the web. Hence, their performance also becomes an important factor in determining which web service to use. In this document, I developed and evaluated a RESTful architecture based web service application as against a traditional SOAP protocol based web service.

The evaluation and comparison of performance showed the benefits of using a RESTful web service and that it is a better alternative over the conventional SOAP based service. Advantages includes lightweight, less message size, self-descriptive with low overhead and high flexibility and less response time. The figure below(Fig. 7) shows a graph of trend of web services being used from year 2005 to year 2011. Therefore, RESTful web services are a much preferred solution for developing web services on a majority of web application implementations.

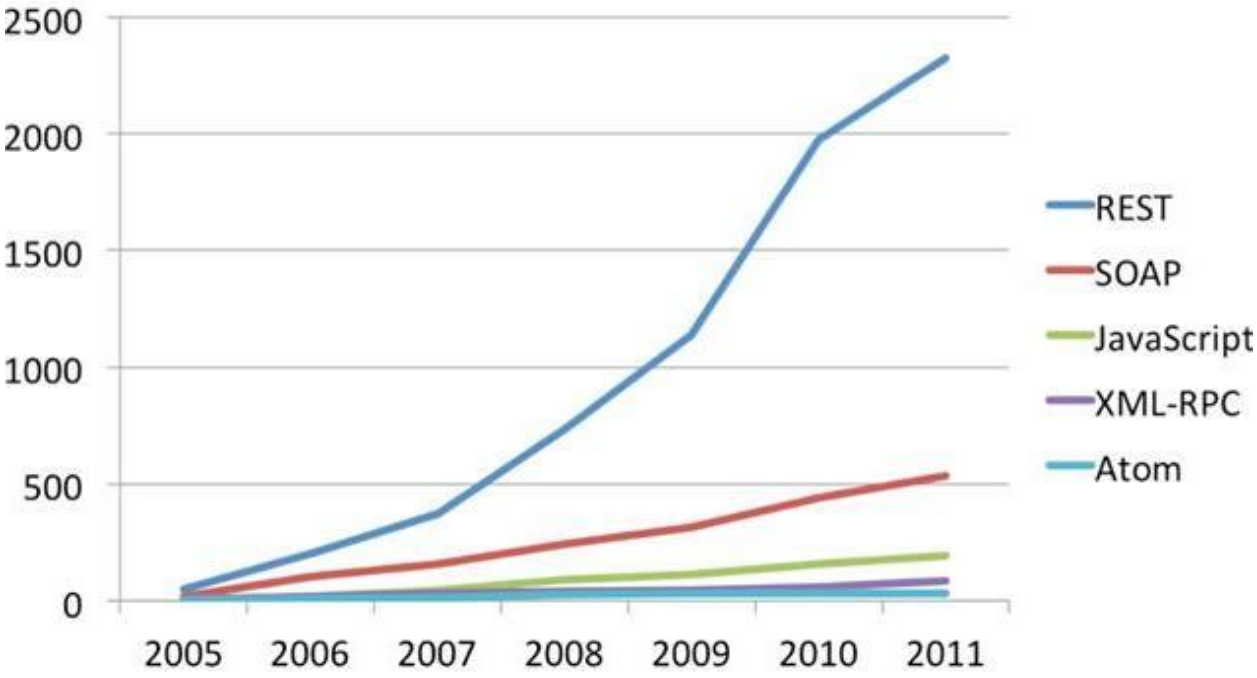


Fig 7: Rest VS SOAP Graph (source : infoq.com)

REFERENCES

[1] Tekli,J.M.;Damiani,E.;Chbeir,R.;Gianini,G.”SOAP Processing Performance and Enhancement” Services Computing, IEEE Transactions on 10.1109/TSC.2011.11 2012 , Page(s): 387 – 403

[2] Hatem Hamad, Motaz Saad, and Ramzi Abed “Performance Evaluation of RESTful Web Services for Mobile Devices” Computer Engineering Department, Islamic University of Gaza, Palestine International Arab Journal of e-Technology, Vol. 1, No. 3, January 2010.

[3] Fielding R., “Architectural Styles and the Design of Network-based Software Architectures," PhD Dissertation, University of California, Irvine, California, USA, 2000.

[4] Otavio Freitas Ferreira Filho, Maria Alice Grigas Varella Ferreira, “Semantic Web Services: A Restful Approach”, IADIS International Conference 2009.

[5] Erik Wilde, Cesare Pautasso, “REST: From Research to Practice” , Springer Publications, Page(s) 21 - 60

[6] Fatna Belqasmi, Jagdeep Singh, Suhib Younis Bani Melhem, and Roch H. Glitho Concordia “SOAP-Based vs. RESTful Web Services A Case Study for Multimedia Conferencing” University Published by the IEEE Computer Society 1089-7801/12/$31.00 © 2012 IEEE.

[7] Fatna Belqasmi and Roch Glitho, Concordia University Chunyan Fu, Tekelec “RESTful Web Services for Service Provisioning in Next-Generation Networks: A Survey “ 0163-6804/11/$25.00 © 2011 IEEE IEEE Communications Magazine • December 2011

[8] Hadley, M. (2009). *Web Application Description Language (WADL)*. Retrieved February 25, 2009, from Web Application Description Language: https://wadl.dev.java.net/wadl20090202.pdf.

[9] http://searchsoa.techtarget.com/essentialguide/Guide-When-and-how-to-use-REST#guideSection2

[10] javabrains.koushik.org

[11] http://www.infoq.com/articles/rest-soap-when-to-use-each

[12] www.service-architecture.com

[13] http://www.soapui.org/testing-dojo/world-of-api-testing/soap-vs--rest-challenges.html

[14] www.programmableweb.com

[15] www.infoq.com