International Journal of Advanced Information Science and Technology (IJAIST) ISSN: 2319:268
Vol.3, No.4, April 2014 DOI:10.15693/ijaist/2014.v3i4.13-16

# RESTful Web Services

Ms. Asnika S.
II M.Tech, CN&E,
N.M.A.M.Institute of Technology,

Mr. Vasudeva Pai
Asst.Professor, IS&E,
N.M.A.M.Institute of Technology,

Mr. Sudhindra R.
C.T.O., M/s Avishkaar Techno-Solutions Private Limited, B'lore.

## ABSTRACT

The networks today are consistent, ubiquitous and are independent of access, technology and devices used. RESTful web services are emerging as a technology for service provisioning. SOAP-based web services are now being replaced by RESTful services. This paper demonstrates on the concept & significance of RESTful web services.

## Keywords

**REST** - Representational state transfer

**SOAP** - Simple Object Access Protocol

**HTTP** - Hypertext Transfer Protocol

## 1. INTRODUCTION

REST is an architectural style consisting of constraints applied to components, connectors and data elements, within a distributed hypermedia system. REST architectural style is applied to the development of web services by replacing the distributed-computing specifications such as SOAP.

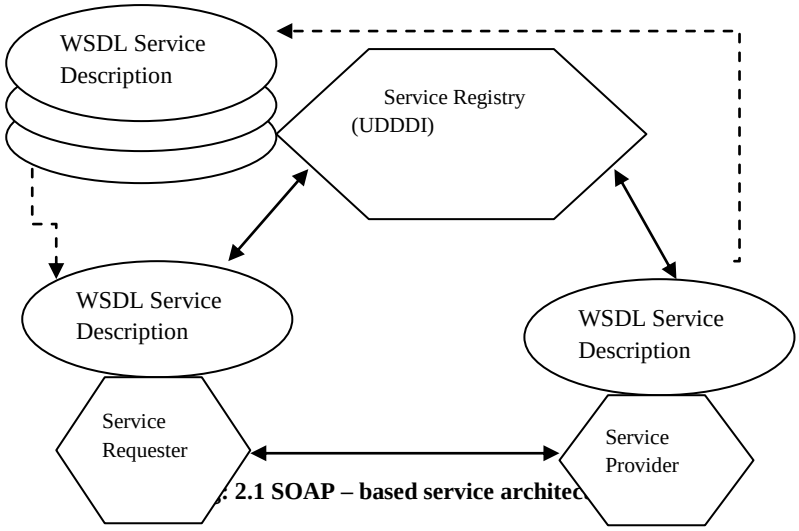
SOAP-based web services have few URIs and many custom methods. SOAP is all about services where as in REST there are many resources hence URIs but few fixed methods allowing users/programmers with no confusions. Also REST is resource-oriented architecture.

## 2. REST OVERVIEW

In this section SOAP – based web services is introduced along with its characteristics. Then concept of RESTful services is discussed along with its design principles, advantages.

The SOAP-based web service architecture defines three entities: service provider, service registry, and service requester **(Fig. 2.1).** The service provider creates a SOAP-based web service and publishes the service description in the service registry. The service requester finds the service by querying the service registry, retrieves the service description, and then uses the description to bind to the service implementation and start interacting with it. The service registry aims at the online discovery of services. However, it is rarely used today because most requesters have prior knowledge of existing services, thanks to offline business agreements. The communications (operations) among the three web service entities are based on XML and use the Simple Object Access Protocol. SOAP messages are commonly exchanged over HTTP, even though other bindings are possible. The service descriptions are published using the Web Services Description Language (WSDL). WSDL provides information on how to use a web service, including a description of the service operations and binding information.

The most commonly used service registry for SOAP based web services is the Universal Description, Discovery and Integration (UDDI) registry. The UDDI specifications define a set of application programming interfaces (APIs) for both publication and discovery. The operations exposed by a SOAP-based web service (e.g., create Conference or add Participant in the case of a SOAP-based web service for conferencing) are defined by the service provider, and each provider can define its own operations (i.e., an operation's name, parameters, and behavior). However, SOAP-based web services can be standardized as a means to increase interoperability, as with the Parlay-X multimedia conferencing web service. The list of exposed operations is then included in the service description.




**. 2.1 SOAP – based service architec**

**Some of the characteristics of SOAP are:**

- SOAP builds an XML protocol on top of HTTP or sometimes TCP/IP.
- SOAP is a successor of XML-RPC and is very similar, but describes a standard way to communicate.
- Several programming languages have native support for SOAP, you typically feed it a web service URL and you can call its web service functions without the need of specific code.
- Binary data that is sent must be encoded first into a format such as base64 encoded.

## 2.1 REST PRINCIPLE

REST is an architectural style consisting of constraints applied to components, connectors and data elements, within a distributed hypermedia system. REST

architectural style is applied to the development of web services by replacing the distributed-computing specifications such as SOAP.

REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. The term *representational state transfer* was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation at UC Irvine.

REST describes a set of architectural principles by which data can be transmitted over a standardized interface (such as HTTP). REST does not contain an additional messaging layer and focuses on design rules for creating stateless services. A client can access the resource using the unique URI and a representation of the resource is returned. With each new resource representation, the client is said to transfer state. 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.

**Some of the many advantages are:**

- REST is almost always going to be faster.
- REST is much more lightweight and can be implemented using almost any tool, leading to lower bandwidth and shorter learning curve. However, the clients have to know what to send and what to expect.
- The RESTful Web services are completely stateless. This can be tested by restarting the server and checking if the interactions are able to survive.
- Restful services provide a good caching infrastructure over HTTP GET method (for most servers). This can improve the performance, if the data the Web service returns is not altered frequently and not dynamic in nature.
- REST is particularly useful for restricted-profile devices such as mobile and PDAs for which the overhead of additional parameters like headers and other SOAP elements are less.
- REST services are easy to integrate with the existing websites and are exposed with XML so the HTML pages can consume the same with ease.

## 2.2 REST vs SOAP

***• SOAP-based web service***

> Few URIs (nouns), many custom methods (verbs). Example **musicPort.getRecordings ("beatles")**

> Uses HTTP as transport for SOAP messages

***• RESTful web service***

> Many resources (nouns), few fixed methods (verbs). Example

**GET /music/artists/beatles/recordings**

> HTTP is the protocol.

**• SOAP Service and REST Resource**

> SOAP based web services is about services

Stock quote service quoteService.purchase ("sunw", 2000, 6.0f);

**• REST is Resource-Oriented Architecture**

> Stock quote resource

> Resources are manipulated by exchanging representations, Eg. **purchasing** stock

# 3. REST DESIGN

With the upcoming technology we access web not only from computer but also from phones. There are containers which stores resources and using these resources various services are accessed. Today we access web through URLs. Various services are accessed through URLs.

**RESTful services have the following design characteristics:**

• RESTful services have a uniform interface through well known HTTP methods

> GET, POST, PUT, and DELETE

• REST-based architectures are built from resources (pieces of information) that are uniquely identified by URIs

• RESTful services are stateless

> Each request from client to server must contain all the Information necessary to understand the request In REST system, resources are manipulated through the exchange of "representations" of the resources

> For example, a purchase order resource is represented by an XML or JSON document.

> In a RESTful purchasing system, each purchase order is made through a combination of HTTP POST method with XML document, which represents the order, sent to a unique URI In REST system, communication occurs primarily through the transfer of representations of resources

> REpresentational State Transfer. The basic design architecture is shown below in the figure:

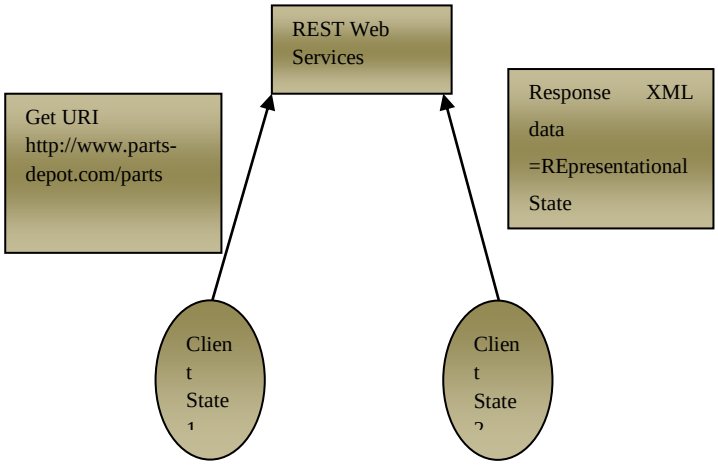



**Figure 3.1: Design Architecture**

## 3.1 Representational State Transfer

In REST everything has an ID. Everything as in every resources used as ID and using these URI the resource is accessed and thus the service. Some of the examples are shown below:

ID is a URI

http://example.com/widgets/foo

http://example.com/customers/bar

Since URI is used to access the resources, each URI must depict in meaningful sense. As an example considers the URI below and the meaning it has:

***Example URI***

http://www.sun.com/servers/blades/t6300

Resource Collection ⟶ servers

Name ⟶ blades

Primary key ⟶ t6300

As discussed REST allows multiple representation formats which helps the user as well as programmer greatly. REST offers data in variety of formats like:

> XML

> JSON

> (X)HTML

In RESTful services content negotiation can be done in two forms

> Accept header

**GET /foo**

**Accept: application/json**

> URI-based

**GET /foo.json**

Let us understand the method how a REST request is made. The following are the steps in making a REST request. Resources (nouns) used for requesting the service is identified by a URI, for example:

> http://ww

w.parts-depot.com/parts

Methods (verbs) used to manipulate the nouns are

> Small fixed set: GET, POST, PUT, and DELETE

Representation is how you view the State and the format can be in XML, JSON etc. The data and state are transferred between the client and server. The verbs mentioned above must be used to exchange the application state and representation. Once we have understood the REST request let us see HTTP request as REST as the protocol used and universal way is HTTP.

***HTTP Request/Response As REST***

***Request***

GET /music/artists/beatles/recordings HTTP/1.1

Host: media.example.com

Accept: application/xml

***Response***

HTTP/1.1 200 OK

Date: Tue, 08 May 2007 16:41:58 GMT

Server: Apache/1.3.6

Content-Type: application/xml; charset=UTF-8

**Representation**

<?xml version="1.0"?>

<recordings xmlns="…">

<recording>…</recording>

Once we have understood the request and response format let us now understand the operation performed using RESTful services. **Following operations are used:**

- GET: retrieve whatever information (in the form of an entity) is identified by the Request-URI. Retrieve representation, shouldn't result in data modification.
- POST: request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI. Annotation of existing resources, extending a database through an append operation, posting a message to a bulletin board, or group of articles. Used to change state at the server in a loosely coupled way (update).
- PUT: requests that the enclosed entity be stored under the supplied Request-URI, create/put a new resource. Its used to set some piece of state on the server.
- DELETE: requests that the origin server deletes the resource identified by the Request-URI.

**A distributed hypermedia architect has only three fundamental options:**

- Render the data where it is located and send a fixed-format image to the recipient.
- Encapsulate the data with a rendering engine and send both to the recipient.
- Send the raw data to the recipient along with metadata that describes the data type, so that the recipient can choose their own rendering engine.

**The following table summarizes the REST principles**

**Table1: REST Principles**

| CRUD | REST |
|---|---|

| | | |
|---|---|---|
| **Create** | Put | Initialize the state of a new resource at the given URI |
| **Read** | Get | Retrieve the current state of the resource |
| **Update** | Post | Modify the state of the resource |
| **Delete** | Delete | Clear the resource when it is no longer valid. |

***HTTP Methods: GET***

GET HTTP method is used to retrieve information from the server as requested by the client or the user. The information is retrieved from the URI mentioned. GET method is idempotent meaning should not initiate any state change to it. GET method has the characteristics to be cached.

> ***GET /pictures/flowers***

***HTTP Methods: POST***

POST HTTP method is used to add any new information to the destination mentioned. POST method can be used to add the entity as a subordinate/append to the POSTed resource. For example consider the request below:

> ***POST /pictures/flowers***

Adds the pictures specified in the POSTDATA to the list of pictures

***HTTP Methods: PUT***

PUT HTTP method is used to update information in the destined file. PUT method can be used to create a full new entity, or replace any information when we know the id. Consider for example the request below:

> ***PUT /flowers/123-45678901***

Replaces the representation of the Picture (flower) with an updated version.

***HTTP Methods: DELETE***

DELETE HTTP method is used to remove (logical) an entity. For example consider the code below which says delete a picture in the list of flowers.

> ***DELETE /flower/hibiscus***

Deletes the flower **hibiscus** from the system.

Once we are clear with request and response format, HTTP methods let us briefly view all the modules as a whole. Here we consider a container which contains some items. Below shown are the ways in which methods are used in different scenarios.

***Common Patterns: Container, Item***

Server is in the control of URI path space. To list container contents we use **GET /items**. To add item to container we use **POST /items**. In the above scenario item is used as request. The URI of item returned in HTTP response header is as shown: Location: **http://host/items/itemid.** To read item from the container we use **GET /items/itemid**. To update item in the container we use **PUT/items/itemid**. Here item is used as request. In the similar way to remove an item from the container we use **DELETE /items/itemid**.

## 4. CONCLUSIONS

REST has been widely used outside NGNs. However, several standards bodies are attempting to produce standard specifications for REST based service provisioning in NGNs (e.g., OMA and IETF). Some work has also been done in the area outside standards bodies. RESTful web services meet many NGN service provisioning requirements. They enable easy development and deployment of a wide range of services, support a wide range of terminals (e.g., laptops, cell phones), and allow for service composition through mash ups. However, some issues are still open, such as RESTful web services publication and discovery, resource definition for session-based services, and the provisioning of adequate middleware. RESTful web services do indeed show a great potential for service provisioning in NGNs. Nevertheless, the open issues need to be solved before their full potential can be realized. Referring to some of the web services the articles, we conclude that there is a gateway to the web technologies called RESTful web services.

## REFERENCES

[1] RESTful Web Services for Service Provisioning in Next-Generation Networks: A Survey. IEEE Communications Magazine. December 2011

[2] S. Vinoski, "RESTful Web Services Development Checklist," *IEEE Internet Computing*, vol. 8, no. 6, 2004, pp. 94–95.

[3] D. Bryson and S. Vinoski, "Build Your Next Web Application with Erlang," *IEEE Internet Computing*, vol. 13, no. 4, 2009, pp. 93–96.

[4] G. Montenegro *et al.*, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," IETF RFC 4944, Sept. 2007.

*[5]* E. Newcomer, *Understanding Web Services: XML, WSDL,*

[6] *SOAP, and UDDI*, AddisonWesley, 2002.

[7] R.T. Fielding, "Architectural Styles and the Design of Network based Software Architectures," doctoral dissertation, Dept. of Information and Computer Science, Univ. of California, Irvine, 2000.

[8] *Parlay X Web Services; Part 12: Multimedia Conference*, 3GPP Global Initiative, 2010.

[9] F. Belqasmi et al., "RESTful Web Services for Service Provisioning in Next Generation Networks: A Survey," *IEEE Comm.*, vol. 49, no. 12, 2011, pp. 66–73.