REST API Modeling Languages - A Developer's Perspective

Vijay Surwase
Student
Department of Computer Engineering
Pune Institute of Computer Technology, Pune

Abstract

Service Oriented Architecture (SOA) is generally used while developing enterprise software solutions. This provides flexibility to develop different business modules as services there by providing scalability and reliability. So application becomes loosely coupled. SOAP and REST are two famous approaches of web services. SOAP (Simple Object Access Protocol) is protocol while REST (Representational State Transfer) is architectural style in which services are accessed in form of Resources and generally deployed over web communicates through standard HTTP protocol. Modeling Languages are used to express knowledge/information about system that we are going to develop. For REST service Various Modeling languages / framework are present in REST API worlds. RAML, Swagger and API Blueprint are now a day widely used in development of REST APIs. The paper contains survey of existing modeling languages and guide for choosing best modeling language for REST API development.

Keywords: Modeling Languages, Web services, REST, Service Oriented Architecture (SOA)

I. INTRODUCTION

REST stands for Representational State transfer. This term was first coined by Roy Fielding[17]. REST refers to architectural Style. REST architectural style is gaining popularity but there is lot of debate and growing concern about modeling of REST web services (REST API). The REST architectural style describes six constraints. These constraints, applied to the architecture, were originally communicated by Roy Fielding in his doctoral dissertation and define the basis of RESTful-style. Uniform Interface, Stateless, Cacheable, Client-Server, Layered System, Code on Demand are the constraints imposed by REST architecture[17].

Uniform Interface: It defines the interface between service consumer and service provider. It simplifies and decouples the architecture, which enables independence of each part. Resource-Based Individual resources are identified in requests using URIs as resource identifiers. Representations of resources are logically separated from each other. Client retrieve representation of resource along with some metadata information such as tags, timestamps, etc. This information guides client for performing further operations on resource.

Hypermedia as the Engine of Application State (HATEOAS) is used for guiding Application on server side. API consumer sends its state through request body, query parameters, headers and URI. Service provider sends its state of application through response body, codes, and response headers. This is called technically as hypermedia (or hyperlinks within hypertext). Apart from that, HATEOS means in service response links are attached along with response content in the returned body so that client can get information to form further the URIs for further operation on resources and for other objects in collection of resources[2]. The uniform interface that any REST services must provide is fundamental to its design.

Stateless As REST deals with state transfer, statelessness is power present in REST. It implies is that the state to control the request is resides inside the request itself, whether as part of the URI, request parameters, body, or headers. The URI uniquely identifies the resource and the body involves the state (or state exchange) of that useful resource[1]. Server gets to know about the state of client from this request. Based on that response is generated with suitable state in that and communicated to client.

Cacheable As on the world large web, clients can cache responses. Responses have to hence, implicitly or explicitly, outline themselves as cacheable, or no longer, to avert customers reusing stale or inappropriate data in line with extra requests. Good-managed caching partly or thoroughly eliminates some purchaser– server interactions, extra bettering scalability and efficiency [3].

Consumer–server The uniform interface separates customers from servers. This separation of considerations implies that, for instance, purchasers aren't concerned with knowledge storage, which stays inner to every server, in order that the portability of client code is improved. Servers are usually not concerned with the person interface or consumer state, in order that servers will also be easier and extra scalable. Servers and purchasers will also be replaced and developed independently, as long as the interface will not be altered.

Layered procedure API consumer cannot traditionally tell whether or not it is hooked up directly to the top server, or to an middleman along the way[5]. Middleman servers could improve procedure scalability via enabling load-balancing and through delivering shared caches. Layers may additionally implement protection policies.

Code on demand (optional) Servers are ready to temporarily prolong or customise the performance of a consumer by way of transferring good judgment to it that it could actually execute. Examples of this may incorporate compiled add-ons equivalent to Java applets and patron-side scripts such as JavaScript.

II. ABOUT REST API

A REST API describes a set of resources, and a set of operations that can be called on those resources. The operations in a REST API can be called from any HTTP client, including client-side JavaScript code that is running in a web browser[5].
The REST API has a base path, which is similar to a context root. All resources in a REST API are defined relative to its base path. The base path can be used to provide isolation between different REST APIs, as well as isolation between different versions of the same REST API. For example, a REST API can be built to expose a student database over HTTP. The base path for the first version of that REST API could be /studentdb/v1, while the base path for the second version of that REST API could be /studentdb/v2.

Resource	*Description*
/students	*All of the students in the database*
/students/12345	*Student #12345*
/students/12345/orders	*All orders for student #12345*
/students/12345/orders/67890	*Order #67890 for student #12345*

The REST API describes a set of resources. The HTTP client uses a path relative to the base path that identifies the resource in the REST API that the client is accessing. The paths to a resource can be hierarchical, and a well-designed path structure can help a consumer of a REST API understand the resources available within that REST API. The above table lists some example resources for a student database in the REST API. Each resource in the REST API has a set of operations that can be called by an HTTP client. An operation in a REST API has a name and an HTTP method (such as GET, POST, or DELETE). The name of the operation must be unique across all of the resources in that REST API. Additionally, a single resource can have only one operation that is defined for a specific HTTP method[8]. The combination of the path and the HTTP method that are specified by the HTTP client that is making the request is used to identify the resource and operation that is being called.

HTTP Method	*Operation Name*	*Description*
GET	*getStudent*	*Retrieve the student details from the database.*
PUT	*updateStudent*	*Update the student details in the database.*
DELETE	*deleteStudent*	*Delete the student from the database.*

To call the updateStudent operation, the HTTP client makes an HTTP PUT request to /studentdb/v1/students/12345.
Each operation in a REST API can also have a set of parameters that can be used by the HTTP client to pass arguments into the operation. Each parameter must be defined in the definitions for the REST API. Each parameter has a unique name and type.

A. Path Parameters:

These can be used to identify a particular resource. The value of the parameter is passed in to the operation by the HTTP client as a variable part of the URL, and the value of the parameter is extracted from the path for use in the operation. For example, the student ID can be passed in as a path parameter named studentId: /students/{studentId}

B. Query Parameters:

The value of a query parameter is passed in to the operation by the HTTP client as a key value pair in the query string at the end of the URL. As an example, query parameters can be used to pass in a minimum and maximum number of results that should be returned by a particular operation:
/students?min=5&max=20

C. Header Parameters:

The HTTP client can pass header parameters to an operation by adding them as HTTP headers in the HTTP request. As an example, header parameters might be used to pass in a unique identifier that identifies the HTTP client that is calling the operation:

Api-Client-Id: ff6e2c5d-42d5-4026-8f7f-d1e56da7f777

A single operation can define and accept multiple parameters of all three types. Additionally, depending on the HTTP method of the operation, the operation can accept data from the HTTP client in the request body. Operations can also send data back to the HTTP client in the response body.

III. WHAT IS NOT REST

There different rumors about REST APIs. One should able to identify the difference between just web API and REST API. REST is just the architectural design Style and it's not protocol[9]. We can consider REST as design pattern with which we can create architecture for our business problem [10]. REST can be implemented outside Web also and Web can implement architectural style other than REST. Another misunderstanding about HTTP protocol. Any web service defined over HTTP is not REST. And REST can be implemented using HTTP and also from other transport layer protocols. The Richardson Maturity Model [12] defines

way to classify REST web service as per REST requirements satisfiability. It defines different levels based on that we can find out up to what extent given web service is RESTful or not

IV. OVERVIEW OF REST API MODELING LANGUAGES

REST API can be specified in various modeling languages. These Modeling languages may represent the JSON(JavaScript Object Notation) /XML (eXtended Markup Language) / YAML (YAML Ain't Markup Language) format. Different Modeling Languages are present in the API World. Many of them are open source.

A. Swagger:

Swagger is an open specification for defining REST APIs [7].
A Swagger document is the REST API equivalent of a WSDL document for SOAP-based web service. The Swagger document specifies the list of resources that are available in the REST API and the operations that can be called on those resources. The Swagger document also specifies the list of parameters to an operation, including the name and type of the parameters, whether the parameters are required or optional, and information about acceptable values for those parameters. Additionally, the Swagger document can include JSON Schema that describes the structure of the request body that is sent to an operation in a REST API, and the json schema describes the structure of any response bodies that are returned from an operation. A swagger community provides a list of tools that will help developer, tester and reviewer to make API development easy. Below are some swagger Tools.
1) Swagger Editor
Assists in building a Swagger document from a web browser by providing a side-by-side view of the Swagger document and the resulting REST API definitions
2) Swagger UI
Table 1 Modeling Languages and their Comparisons
Allows visualizing and testing a REST API that is defined with Swagger from any web browser. The built-in testing functions allow specifying the inputs to an operation that is defined in that REST API, call that operation from the web browser, and inspect the results of calling that operation.
3) Swagger Codegen
Generates a Software Development Kit (SDK) in various languages, including Java™, Objective-C, PHP, and Python, from a Swagger document for a REST API [7]. The resulting SDK can be used to embed calls to the operations in that REST API into a software program that was written in one of the supported languages, without having to handle the underlying HTTP transport.

B. RAML:

RAML (RESTful API Modeling Language) provides a structured, unambiguous format for describing a RESTful API. It allows you to describe API; the endpoints, the HTTP methods to be used for each one, any parameters and their format, what can expected by way of a response and more.[11] RESTful API Modeling Language (RAML) makes it easy to manage the whole API lifecycle from design to sharing. It's concise - only write what we need to define - and reusable. It is machine readable API design that is actually human friendly.
RAML is the only spec designed to encompass the full API lifecycle in a human readable format with code and design pattern reuse. With RAML we can truly design, build, test, document, and share your API all with one spec.

C. API Blueprint:

API Blueprint is simple and accessible to everybody involved in the API lifecycle. Its syntax is concise yet expressive. With API Blueprint you can quickly design and prototype APIs to be created or document and test already deployed mission-critical APIs. API Blueprint is completely open sourced under the MIT license. Its future is transparent and open. API Blueprint doesn't need a closed work group. Instead it uses the RFC process similar to Rust language or Django Enhancement Proposal RFC processes. API Blueprint is built to encourage dialogue and collaboration between project stakeholders, developers and customers at any point in the API lifecycle. At the same time, the API Blueprint tools provide the support to achieve the goals be it API development, governance or delivery.

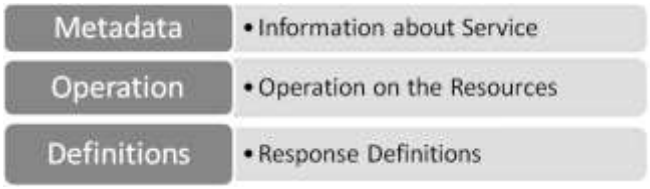

Fig. 1: Generalized Structure of REST API

Table – 1
Modeling Languages and their Comparisons

Name	*Sponsor*	*Format*	*Open Source*	*Code generation (client)*	*Code generation (server)*

RAML	*MuleSoft*	*YAML*	*Yes*	*limited*	*Yes*
API Blueprint	*Apiary*	*Markdown*	*Yes*	*No*	*Yes*
OpenAPI	*Reverb (company)*	*JSON*	*Yes*	*Yes*	*Yes*

Figure 1 is Generalized Structure of REST APIs [15][8]. Generally each API can be divided into below shown three divisions that are Metadata, Operation, and Definitions [13],[9]. Metadata Corresponds to the information about resource such as version of service, description and name of service, Contact information, Base URI etc. Operation section holds information about resource URIs and verbs to specify operation on resources. It also includes response information of REST operation. Definition part tells about complete structure of input and output fields required for REST operations.

V. COMPARISON BETWEEN THE LANGUAGES

Swagger project has the largest and most active developer community on github. All these three standards have large industry backing but analyzing them we found that RAML has baking by Mulesoft company which is leader in API industry [15],[16].

RAML and API Blueprint provide best ease of access through websites. It makes user to easily adopt to current service. Swagger provides GUI editor and rendering effectively. It helps a lot for testing and validating of REST API. Swagger Editor helps for visualizing API while writing and eliminate early mistakes in API specifications. RAML also provides ease of use and console for interacting with API for developer. Both API Blueprint and RAML provides ease for building the documentation of your API. Swagger and RAML support most of programming languages present in the market. While choosing the language for the API developer must keep in mind the strength and weakness of each languages and decide what strength they are required and what weakness they can afford. Swagger has largest support from community hence for developer issue it will be easy for resolving them as soon as possible. Open API initiative has been taken to standardize the REST API specification which extends current swagger specification. Developer should choose the Specification based on tool that available and ease of development it affords.

VI. CONCLUSION

In this paper study of different literature has been discussed. We have presenting information about the API and what exactly is the REST is and what it's not. Also discussed in detail about different modeling languages available in API world. Each language has its own specification with some advantages and disadvantages. Each Specification follows some generalized format described above.

REFERENCES

[1] Marek Polk and Irena Holubov,"REST API Management and Evolution Using MDA",C3S2E '15 Proceedings of the Eighth International C* Conference on Computer Science and Software Engineering , 2015-07- 13, Pages 102-109.
[2] Irum Rauf, Anna Ruokonen, Tarja Systa and Ivan Porres,"Modeling a Composite RESTful Web Service with UML", European Conference on Software Architecture - ECSA '10 Proceedings of the Fourth European Conference on Software Architecture, CompanionVolume, 2010-08-23, pages 253-260.
[3] Michael Owonibi and Peter Baumann,"Heuristic Geo Query Decomposition and Orchestration in a SOA", Information Integration and Web-based Applications and Services - iiWAS'10 Proceedings of the 12th International Conference on Information Integration and Web-based Applications and Services, 2010-11-08, 890-894.
[4] Luca Panziera and Flavio De Paoli, "A Framework for Self-descriptive RESTful Services", International World Wide Web Conference - WWW '13 Companion Proceedings of the 22nd International Conference on World Wide Web, 1407-1414, 2013-05-13.
[5] Michael Petychakis, Fenareti Lampathaki and Dimitrios Askounis, "Adding Rules on Existing Hypermedia APIs", International World Wide Web Conference - WWW '15 Companion Proceedings of the 24th International Conference on World Wide Web, 1515-1517, 2015-05-18.
[6] (2016) REST API Modeling language (RAML) website. [Online]. Available: http://www.raml.org/
[7] (2016) Swagger Framework for APIs Website.[Online].Available: http://www.swagger.io/
[8] (2016) Web Application Description Language (WADL) Website. [Online] Available: https://wadl.java.net/
[9] (2016) SOAP Specifications - World Wide Web Consortium Website. [Online] Available: http://www.w3.org/TR/soap12/
[10] (2016) JSON JavaScript Object Notation Website. [Online]. Available: http://www.json.org/
[11] (2016) REST - Semantic Web Standards Website.[Online]. Available:http://www.w3.org/2001/sw/wiki/REST.
[12] Li Li and Wu Chou, "Design and Describe REST API without Violating REST: A Petri Net Based Approach", 2011 IEEE International Conference on Web Services, 508-516, 2011.
[13] Antonio Garrote Hernndez and Mara N.Moreno Garca,"A Formal Definition of RESTful Semantic Web Services" WS-REST 2010, April 26, 2010; Raleigh, NC, USA, 39-46, 2010.
[14] Markku Laitkorpi, Petri Selonen and Tarja Systa,"Towards a Model-Driven Process for Designing ReSTful Web Services" 2009 IEEE International Conference on Web Services, 2009 IEEE, 173-181. W. D. Doyle, "Magnetization reversal in films with biaxial anisotropy," in 1987 Proc. INTERMAG Conf., pp. 2.2-1–2.2-6.
[15] Marios Fokaefs and Eleni Stroulia, "Using WADL Specifications to Develop and Maintain REST Client Applications" 2015 IEEE International Conference on Web Services, 2015 IEEE, 81-89.
[16] Florian Haupt, Frank Leymann and Cesare Pautasso, "A conversation based approach for modeling REST APIs", 2015 12th Working IEEE/IFIP Conference on Software Architecture, 2015 IEEE, 165-175.
[17] (2016) Roy Fielding dissertation report Website.[Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.