IJSTE - Revista Internacional de Ciência, Tecnologia e Engenharia | Volume 2 | Edição 10 | abril de 2016
ISSN (online): 2349-784X

Linguagens de modelagem da API REST - de um desenvolvedor Perspectiva

Vijay Surwase
Aluna
Departamento de Engenharia de Computação
Pune Instituto de Tecnologia da Computação, Pune

Resumo

Arquitetura Orientada a Serviços (SOA) é geralmente usada durante o desenvolvimento de soluções de software empresarial. Isso fornece flexibilidade para desenvolver diferentes módulos de negócios como serviços, fornecendo escalabilidade e confiabilidade. Portanto, a aplicação torna-se fracamente acoplada. SOAP e REST são duas abordagens famosas de serviços web. SOAP (Simple Object Access Protocol) é um protocolo, enquanto REST (Representational State Transfer) é um estilo arquitetônico no qual os serviços são acessados na forma de Recursos e geralmente implantados na Web, por meio do protocolo HTTP padrão. As Linguagens de Modelagem são utilizadas para expressar conhecimentos/informações sobre o sistema que vamos desenvolver. Para o serviço REST Várias linguagens / estrutura de modelagem estão presentes nos mundos da API REST. RAML, Swagger e API Blueprint são hoje amplamente utilizados no desenvolvimento de APIs REST.

O documento contém um levantamento das linguagens de modelagem existentes e um guia para escolher a melhor linguagem de modelagem para o desenvolvimento da API REST.

Palavras-chave: Linguagens de Modelagem, Web services, REST, Service Oriented Architecture (SOA)

I. INTRODUÇÃO

REST significa transferência de estado representacional. Este termo foi cunhado pela primeira vez por Roy Fielding[17]. REST refere-se ao estilo arquitetônico. O estilo arquitetônico REST está ganhando popularidade, mas há muito debate e uma preocupação crescente sobre a modelagem de serviços da Web REST (API REST). O estilo arquitetônico REST descreve seis restrições. Essas restrições, aplicadas à arquitetura, foram originalmente comunicadas por Roy Fielding em sua dissertação de doutorado e definem a base do estilo RESTful. Interface Uniforme, Stateless, Cacheável, Cliente-Servidor, Sistema em Camadas, Code on Demand são as restrições impostas pela arquitetura REST[17].

Interface Uniforme: Define a interface entre o consumidor de serviço e o provedor de serviço. Simplifica e desacopla a arquitetura, o que possibilita a independência de cada parte. Recursos individuais baseados em recursos são identificados em solicitações usando URIs como identificadores de recursos. As representações de recursos são logicamente separadas umas das outras. O cliente recupera a representação do recurso junto com algumas informações de metadados, como tags, carimbos de data/hora, etc. Essas informações orientam o cliente para realizar outras operações no recurso.

A hipermídia como o mecanismo do estado do aplicativo (HATEOAS) é usada para orientar o aplicativo no lado do servidor. O consumidor da API envia seu estado por meio do corpo da solicitação, parâmetros de consulta, cabeçalhos e URI. O provedor de serviços envia seu estado de aplicativo por meio do corpo da resposta, códigos e cabeçalhos de resposta. Isso é chamado tecnicamente de hipermídia (ou hiperlinks dentro do hipertexto). Além disso, HATEOS significa que os links de resposta do serviço são anexados junto com o conteúdo da resposta no corpo retornado, para que o cliente possa obter informações para formar ainda mais os URIs para operação adicional nos recursos e para outros objetos na coleção de recursos [2]. A interface uniforme que qualquer serviço REST deve fornecer é fundamental para seu projeto.

Stateless Como o REST lida com a transferência de estado, a ausência de estado é o poder presente no REST. Isso implica que o estado para controlar a solicitação reside dentro da própria solicitação, seja como parte do URI, parâmetros de solicitação, corpo ou cabeçalhos. O URI identifica exclusivamente o recurso e o corpo envolve o estado (ou troca de estado) desse recurso útil[1]. O servidor fica sabendo sobre o estado do cliente a partir desta solicitação. Com base nessa resposta, é gerada com o estado adequado e comunicada ao cliente.

Armazenável em cache Como na grande rede mundial de computadores, os clientes podem armazenar em cache as respostas. As respostas devem, portanto, implícita ou explicitamente, definir-se como armazenáveis em cache, ou não mais, para evitar que os clientes reutilizem dados obsoletos ou inadequados de acordo com solicitações adicionais. Um bom cache gerenciado elimina parcial ou completamente algumas interações comprador-servidor, melhorando ainda mais a escalabilidade e a eficiência [3].

Consumidor-servidor A interface uniforme separa os clientes dos servidores. Esta separação de considerações implica que, por exemplo, os compradores não se preocupem com o armazenamento de conhecimento, que fica dentro de cada servidor, para que a portabilidade do código do cliente seja melhorada. Os servidores geralmente não se preocupam com a interface do usuário ou estado do consumidor, para que os servidores também sejam mais fáceis e mais escaláveis. Servidores e compradores também serão substituídos e desenvolvidos de forma independente, desde que a interface não seja alterada.

O consumidor da API de procedimento em camadas tradicionalmente não pode dizer se está ou não conectado diretamente ao servidor principal ou a um intermediário ao longo do caminho [5]. Os servidores intermediários podem melhorar a escalabilidade do procedimento por meio da habilitação do balanceamento de carga e do fornecimento de caches compartilhados. As camadas também podem implementar políticas de proteção.

Código sob demanda (opcional) Os servidores estão prontos para prolongar temporariamente ou personalizar o desempenho de um consumidor por meio da transferência de um bom julgamento para que ele realmente possa executar. Exemplos disso podem incorporar complementos compilados equivalentes a applets Java e scripts do lado do usuário, como JavaScript.

II. SOBRE API REST

Uma API REST descreve um conjunto de recursos e um conjunto de operações que podem ser chamadas nesses recursos. As operações em uma API REST podem ser chamadas de qualquer cliente HTTP, incluindo código JavaScript do lado do cliente que está sendo executado em um navegador da Web[5].
A API REST tem um caminho base, que é semelhante a uma raiz de contexto. Todos os recursos em uma API REST são definidos em relação ao seu caminho base.
O caminho base pode ser usado para fornecer isolamento entre diferentes APIs REST, bem como isolamento entre diferentes versões da mesma API REST. Por exemplo, uma API REST pode ser criada para expor um banco de dados de alunos por HTTP. O caminho base para a primeira versão dessa API REST pode ser /studentdb/v1, enquanto o caminho base para a segunda versão dessa API REST pode ser /studentdb/v2.

Recurso /	*Descrição*
alunos	*Todos os alunos no banco de dados*
/alunos/12345	*Aluno nº 12345*
/alunos/12345/pedidos	*Todos os pedidos para o aluno nº*
12345 /students/12345/orders/67890	*Pedido nº 67890 para o aluno nº 12345*

A API REST descreve um conjunto de recursos. O cliente HTTP usa um caminho relativo ao caminho base que identifica o recurso na API REST que o cliente está acessando. Os caminhos para um recurso podem ser hierárquicos e uma estrutura de caminho bem projetada pode ajudar um consumidor de uma API REST a entender os recursos disponíveis nessa API REST. A tabela acima lista alguns recursos de exemplo para um banco de dados de alunos na API REST. Cada recurso na API REST possui um conjunto de operações que podem ser chamadas por um cliente HTTP. Uma operação em uma API REST tem um nome e um método HTTP (como GET, POST ou DELETE). O nome da operação deve ser exclusivo em todos os recursos nessa API REST. Além disso, um único recurso pode ter apenas uma operação definida para um método HTTP específico[8]. A combinação do caminho e do método HTTP especificados pelo cliente HTTP que está fazendo a solicitação é usada para identificar o recurso e a operação que está sendo chamada.

Nome da operação do método HTTP		*Descrição*
GET	*getStudent updateStudent*	*Recupere os detalhes do aluno do banco de dados.*
COLOCAR	*deleteStudent*	*Atualize os detalhes do aluno no banco de dados.*
EXCLUIR		*Exclua o aluno do banco de dados.*

Para chamar a operação updateStudent, o cliente HTTP faz uma solicitação HTTP PUT para /studentdb/v1/students/12345.
Cada operação em uma API REST também pode ter um conjunto de parâmetros que podem ser usados pelo cliente HTTP para passar argumentos para a operação. Cada parâmetro deve ser definido nas definições para a API REST. Cada parâmetro tem um nome e tipo exclusivos.

A. Parâmetros do caminho:

Estes podem ser usados para identificar um recurso específico. O valor do parâmetro é passado para a operação pelo cliente HTTP como uma parte variável da URL e o valor do parâmetro é extraído do caminho para uso na operação. Por exemplo, a ID do aluno pode ser passada como um parâmetro de caminho chamado studentId: /students/{studentId}

B. Parâmetros de consulta:

O valor de um parâmetro de consulta é passado para a operação pelo cliente HTTP como um par de valores-chave na string de consulta no final da URL. Como exemplo, os parâmetros de consulta podem ser usados para passar um número mínimo e máximo de resultados que devem ser retornados por uma determinada operação: /students?min=5&max=20

C. Parâmetros do cabeçalho:

O cliente HTTP pode passar parâmetros de cabeçalho para uma operação adicionando-os como cabeçalhos HTTP na solicitação HTTP. Por exemplo, os parâmetros do cabeçalho podem ser usados para passar um identificador exclusivo que identifica o cliente HTTP que está chamando a operação: Api-Client-Id: ff6e2c5d-42d5-4026-8f7f-d1e56da7f777 Uma única operação pode definir e aceitar vários parâmetros de todos os três tipos. Além disso, dependendo do método HTTP da operação, a operação pode aceitar dados do cliente HTTP no corpo da solicitação. As operações também podem enviar dados de volta ao cliente HTTP no corpo da resposta.

III. O QUE NÃO É DESCANSO

Existem diferentes rumores sobre APIs REST. Deve-se identificar a diferença entre apenas API da Web e API REST. REST é apenas o estilo de design arquitetônico e **não é** protocolo[9]. Podemos considerar REST como um padrão de projeto com o qual podemos criar arquitetura para nosso problema de negócio [10]. O REST também pode ser implementado fora da Web e a Web pode implementar um estilo arquitetônico diferente do REST. Outro mal-entendido sobre o protocolo HTTP. Qualquer serviço da Web definido por HTTP não é REST. E o REST pode ser implementado usando HTTP e também de outros protocolos da camada de transporte. O Modelo de Maturidade de Richardson [12] define

maneira de classificar o serviço da Web REST de acordo com a satisfação dos requisitos REST. Ele define diferentes níveis com base em que podemos descobrir até que ponto determinado serviço da Web é RESTful ou não

4. VISÃO GERAL DAS LINGUAGENS DE MODELAGEM DA API REST

A API REST pode ser especificada em várias linguagens de modelagem. Essas linguagens de modelagem podem representar o formato JSON (JavaScript Object Notation) /XML (eXtended Markup Language) / YAML (YAML Ain't Markup Language). Diferentes linguagens de modelagem estão presentes no API World. Muitos deles são de código aberto.

A. *Arrogância:*

Swagger é uma especificação aberta para definir APIs REST [7].
Um documento Swagger é o equivalente da API REST de um documento WSDL para serviço da Web baseado em SOAP. O documento Swagger especifica a lista de recursos disponíveis na API REST e as operações que podem ser chamadas nesses recursos.
O documento Swagger também especifica a lista de parâmetros para uma operação, incluindo o nome e o tipo dos parâmetros, se os parâmetros são obrigatórios ou opcionais e informações sobre valores aceitáveis para esses parâmetros. Além disso, o documento Swagger pode incluir o esquema JSON que descreve a estrutura do corpo da solicitação que é enviada para uma operação em uma API REST, e o esquema json descreve a estrutura de quaisquer corpos de resposta que são retornados de uma operação. Uma comunidade swagger fornece uma lista de ferramentas que ajudarão o desenvolvedor, testador e revisor a facilitar o desenvolvimento de APIs. Abaixo estão algumas ferramentas de swagger.

1) Editor Swagger

Auxilia na criação de um documento Swagger a partir de um navegador da Web, fornecendo uma exibição lado a lado do documento Swagger e as definições de API REST resultantes *Swagger UI* Tabela 1 Linguagens de modelagem e suas comparações Permite visualizar e testar uma API REST definida com Swagger de qualquer
2) navegador da web. As funções de teste integradas permitem especificar as entradas para uma operação definida nessa API REST, chamar essa operação a partir do navegador da web e inspecionar os resultados da chamada dessa operação.

3) Swagger Codegen

Gera um Kit de Desenvolvimento de Software (SDK) em várias linguagens, incluindo Java™, Objective-C, PHP e Python, a partir de um documento Swagger para uma API REST [7]. O SDK resultante pode ser usado para incorporar chamadas para as operações nessa API REST em um programa de software que foi escrito em uma das linguagens suportadas, sem ter que lidar com o transporte HTTP subjacente.

B. *RAML:*

RAML (RESTful API Modeling Language) fornece um formato estruturado e inequívoco para descrever uma API RESTful. Ele permite que você descreva a API; os endpoints, os métodos HTTP a serem usados para cada um, quaisquer parâmetros e seu formato, o que pode ser esperado por meio de uma resposta e muito mais. [11] RESTful API Modeling Language (RAML) facilita o gerenciamento de todo o ciclo de vida da API, desde o design até o compartilhamento. É conciso - escreva apenas o que precisamos definir - e reutilizável. É um design de API legível por máquina que é realmente amigável para humanos.

RAML é a única especificação projetada para abranger todo o ciclo de vida da API em um formato legível por humanos com reutilização de padrão de design e código. Com o RAML, podemos realmente projetar, construir, testar, documentar e compartilhar sua API, tudo com uma especificação.

C. *Projeto de API:*

O API Blueprint é simples e acessível a todos os envolvidos no ciclo de vida da API. Sua sintaxe é concisa, mas expressiva. Com o API Blueprint, você pode projetar e prototipar rapidamente APIs a serem criadas ou documentar e testar APIs de missão crítica já implantadas.
API Blueprint é totalmente open source sob a licença MIT. Seu futuro é transparente e aberto. O API Blueprint não precisa de um grupo de trabalho fechado. Em vez disso, ele usa o processo RFC semelhante à linguagem Rust ou aos processos RFC da proposta de aprimoramento do Django.
O API Blueprint foi criado para incentivar o diálogo e a colaboração entre as partes interessadas, desenvolvedores e clientes do projeto em qualquer ponto do ciclo de vida da API. Ao mesmo tempo, as ferramentas API Blueprint fornecem o suporte para atingir os objetivos, seja desenvolvimento, governança ou entrega de API.

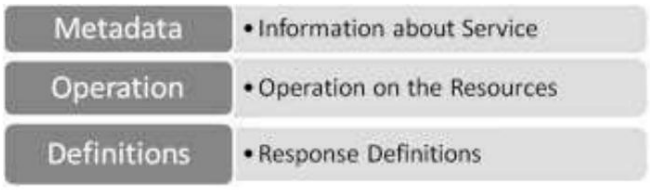

Fig. 1: Estrutura Generalizada da API REST

Tabela - 1

Linguagens de modelagem e suas comparações

Nome	*Patrocinador*	*Formato Open Source Geração de código (cliente) Geração de código (servidor)*			

RAML	*MuleSoft*	*YAML*	*Sim*	*limitado*	*Sim*
API Blueprint	*Apiário*	*Remarcação*	*Sim*	*Não*	*Sim*
OpenAPI Reverb (empresa) JSON			*Sim*	*Sim*	*Sim*

A Figura 1 é a Estrutura Generalizada das APIs REST [15][8]. Geralmente, cada API pode ser dividida em três divisões mostradas abaixo que são Metadados, Operação e Definições [13],[9]. Metadados Corresponde às informações sobre o recurso, como versão do serviço, descrição e nome do serviço, informações de contato, URI base, etc. A seção de operação contém informações sobre URIs de recursos e verbos para especificar a operação nos recursos. Também inclui informações de resposta da operação REST. A parte de definição informa sobre a estrutura completa dos campos de entrada e saída necessários para as operações REST.

V. COMPARAÇÃO ENTRE AS LÍNGUAS

O projeto Swagger tem a maior e mais ativa comunidade de desenvolvedores no github. Todos esses três padrões têm grande respaldo da indústria, mas analisando-os, descobrimos que o RAML foi desenvolvido pela empresa Mulesoft, que é líder na indústria de APIs [15],[16].

RAML e API Blueprint fornecem a melhor facilidade de acesso por meio de sites. Faz com que o usuário adote facilmente o serviço atual. Swagger fornece editor GUI e renderização de forma eficaz. Ajuda muito para testar e validar a API REST. O Swagger Editor ajuda a visualizar a API enquanto escreve e elimina erros iniciais nas especificações da API. RAML também fornece facilidade de uso e console para interagir com a API para o desenvolvedor. Tanto o API Blueprint quanto o RAML facilitam a criação da documentação de sua API. Swagger e RAML suportam a maioria das linguagens de programação presentes no mercado. Ao escolher o idioma para o desenvolvedor da API, deve-se ter em mente os pontos fortes e fracos de cada idioma e decidir quais pontos fortes são necessários e quais pontos fracos podem suportar. O Swagger tem o maior suporte da comunidade, portanto, para o problema do desenvolvedor, será fácil resolvê-lo o mais rápido possível. A iniciativa da API aberta foi tomada para padronizar a especificação da API REST, que estende a especificação atual do swagger. O desenvolvedor deve escolher a Especificação com base na ferramenta disponível e na facilidade de desenvolvimento que ela oferece.

VI. CONCLUSÃO

Neste artigo, o estudo de diferentes literaturas foi discutido. Apresentamos informações sobre a API e o que exatamente é **REST e o que não é. Também discutido em detalhes sobre as diferentes** linguagens de modelagem disponíveis no mundo da API. Cada linguagem tem sua própria especificação com algumas vantagens e desvantagens. Cada Especificação segue algum formato generalizado descrito acima.

REFERÊNCIAS

[1] **Marek Polk e 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, páginas 102-109.

[2] **Irum Rauf, Anna Ruokonen, Tarja Systa e 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, páginas 253-260.

[3] **Michael Owonibi e 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 , 08/11/2010, 890-894.

[4] Luca Panziera e Flavio **De Paoli, “A Framework for Self-descriptive RESTful Services”, International World Wide Web Conference** - **WWW '13** Anais complementares da 22ª Conferência Internacional sobre a World Wide Web, 1407-1414, 2013-05-13.

[5] Michael Petychakis, Fenareti Lampathaki **e Dimitrios Askounis, “Adicionando Regras para APIs de Hipermídia Existentes”, Conferência Internacional da World Wide Web** - **WWW '15 Companion Proceedings of the 24th International Conference on World Wide Web,** 1515-1517, 2015-05-18.

[6] (2016) Site da linguagem de modelagem REST API (RAML). [On-line]. Disponível: http://www.raml.org/ [7] (2016) Swagger Framework for APIs Website.[Online].Disponível: http://www.swagger.io/ [8] (2016) Web Application Description Language (WADL) Site. [Online] Disponível: https://wadl.java.net/ [9] (2016) Especificações SOAP - Site do World Wide Web Consortium. [Online] Disponível: http://www.w3.org/TR/soap12/ [10] (2016) JSON JavaScript Object Notation Website. [On-line]. Disponível: http://www.json.org/ [11] (2016) REST - Semantic Web Standards Website.[Online]. Disponível: http://www.w3.org/2001/sw/wiki/REST.

[12] **Li Li e Wu Chou, “Projetar e descrever a API REST sem violar o REST: uma abordagem baseada em rede de Petri”, 2011 IEEE International** Conference on Web Services, 508-516, 2011.

[13] **Antonio Garrote Hernndez e Mara N. Moreno Garca, “Uma Definição Formal de RESTful Semantic Web Services”** WS-REST 2010, 26 de abril de 2010; Raleigh, NC, EUA, 39-46, 2010.

[14] **Markku Laitkorpi, Petri Selonen e Tarja Systa, “Rumo a um processo orientado a modelos para projetar serviços da Web ReSTful” 2009** Conferência internacional IEEE sobre serviços da Web, 2009 IEEE, **173-181. WD Doyle, "Reversão de magnetização em filmes com anisotropia biaxial", em 1987 Proc. INTERMAG** Conf., pp. 2.2-1–2.2-6.

[15] **Marios Fokaefs e Eleni Stroulia, “Usando especificações WADL para desenvolver e manter aplicações cliente REST” 2015 IEEE International Conference** em serviços da Web, 2015 IEEE, 81-89.

[16] **Florian Haupt, Frank Leymann e Cesare Pautasso, “Uma abordagem baseada em conversação para modelagem de APIs REST”, 2015 12ª Conferência de Trabalho** IEEE/IFIP sobre Arquitetura de Software, 2015 IEEE, 165-175.

[17] (2016) Site do relatório de dissertação de Roy Fielding. [Online]. Disponível: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.