REST

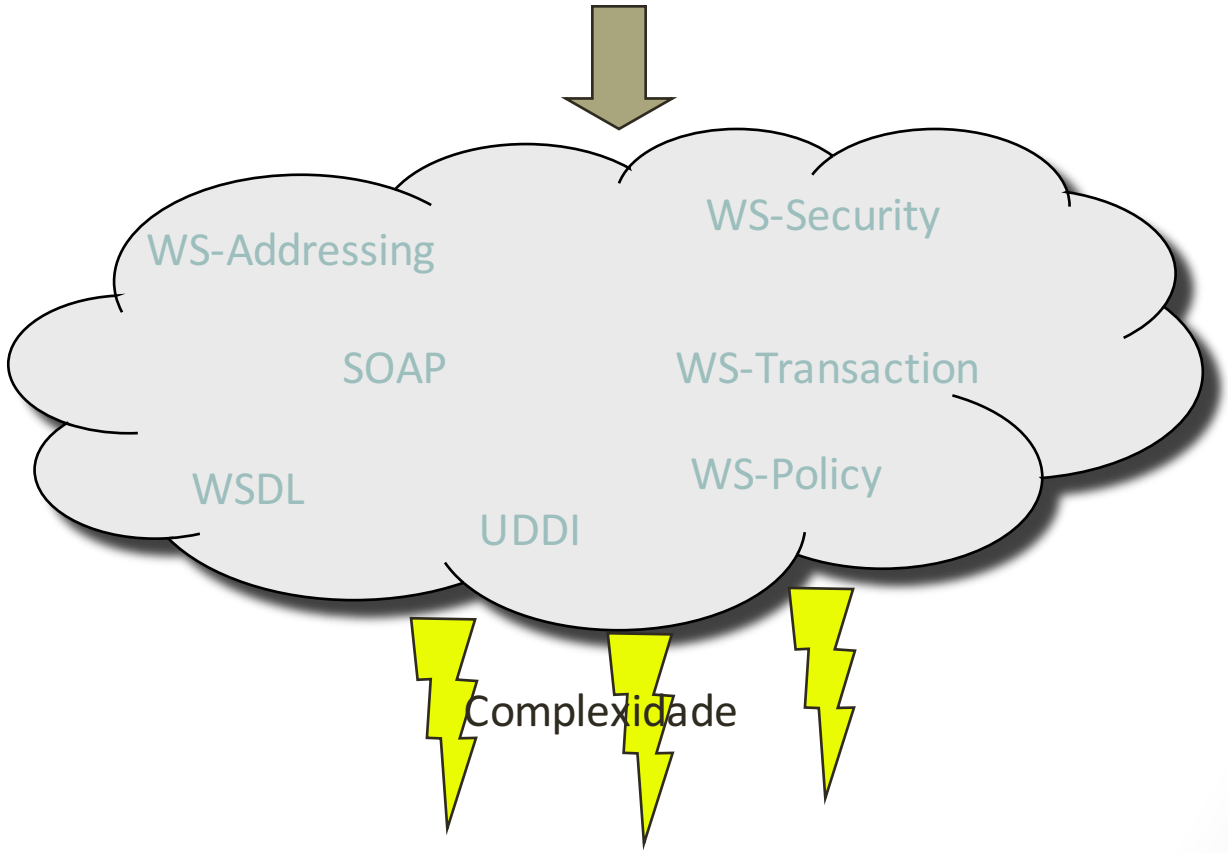
Ricardo Sabatine

Comunicação

- **Qual a forma de comunicação?**
  - Bloqueante, não bloqueante, síncrona ou assíncrona
  - O que é sockets?
- **Qual camada de transporte utilizar?**
  - **HTTP, FTP…**
- **Representação dos Dados?**
  - Linguagem de marcação
  - Documento válido
  - XML Schema, DTD
  - DOM, SAX
- **Web Service**
  - XML-RPC
  - SOAP
  - REST

Sem padrões, difícil vender middleware

Fabricantes padronizam implementações



RESUMO - SOAP

- A arquitetura WS-* é composta por mais de 20 especificações.
- Sendo as especificações base deste conjunto a SOAP, WSDL e UDDI.
  - Além dos citados, existem também os padrões WS-Notification, WS-Addressing, WS-Transfer, WS-Policy, WSSecurity, WSTransaction entre outros.
- Uma das principais reclamações em relação a esse formato diz respeito a esse grande número de especificações que o torna complexo e burocrático, difícil de ser dominado por uma só pessoa.

Padrões WS-* - Problema

- Alguns padrões WS-* (como o SOAP) podem ser usados para implementar as aplicações Remote Procedure Call no HTTP.
  - O que não gostamos é da complexidade desnecessária.
    - Como muita frequência, um programador ou uma empresa produz um Web Services SOAP para um serviço com o qual o velho HTTP comum poderia lidar muito bem.
    - O efeito é que o HTTP é reduzido a um protocolo de transporte para um pesado XML que explica o que “realmente” está ocorrendo.
    - O serviço resultante é muito complexo, impossível de depurar e não funcionará, a menos que seus clientes tenham a configuração exatamente igual à sua.

Padrões WS-* - Problema

- Os Web Services Soap - Vantagem:
  - Ferramentas modernas podem criar um serviço web a partir de seu código com um único clique
  - Provavelmente não importará para você se um serviço web REST for muito mais simples.
  - As ferramentas ocultam toda a complexidade, portanto quem se importa? A largura de banda e a CPU são baratas.
  - Essa atitude é interessante e funciona bem quando você está trabalhando em um grupo homogêneo, fornecendo serviços através de um firewall.

O que é REST?

- REST = REpresentational State Transfer
  - REST ≠ Tecnologia
  - REST ≠ Padrão
  - **REST = Estilo de Arquitetura**
- Um estilo de arquitetura é um conjunto coordenado de restrições arquiteturais que restringe as funções/características dos elementos da arquitetura e permite relações entre esses elementos dentro de qualquer arquitetura que está de acordo com esse estilo.
  - ***Roy Fielding***

REST

- REST é um estilo híbrido derivado de vários estilos de arquiteturas de rede

- REST = LCODC$SS+U

  - Cliente-Servidor
  - STATELESS
  - Interface Uniforme
  - CACHE
  - Code-on-demand

REST – Cliente-Servidor

- Cliente-Servidor

- Separação de responsabilidade
  - A interface de usuário do processamento ou armazenamento de dados feito pelo servidor.

  - Para melhorar a portabilidade da interface do usuário em várias plataformas e do lado servidor melhorar a escalabilidade simplificando os componentes.

  - Permitir a evolução dos componentes de forma independente.
    - Esta restrição isola o cliente das alterações no servidor, uma vez que, entre duas requisições, não há ligação

REST - STATELESS

- Um protocolo cliente/servidor sem estado (STATELESS)
  - Nem o cliente e nem o servidor têm a necessidade de armazenar estados transitórios entre as mensagens de pedido e resposta.
    - Esta restrição induz as propriedades de visibilidade, escalabilidade e confiabilidade.
    - Confiabilidade é melhorada porque isso facilita a tarefa de recuperação de falhas parciais.
      - O cliente poderia receber algumas informações do servidor, e enquanto o primeiro fazia algum processamento, o segundo poderia ser até reiniciado, sem que o cliente perceba nada.

REST - STATELESS

- Servidor sem Estado
  - Visibilidade é melhorada porque todas as informações necessárias para a realização do serviço estão na própria requisição.
  - Escalabilidade é melhorada porque não ter que armazenar o estado entre os pedidos e simplifica ainda mais a aplicação
    - o servidor não precisa gerenciar o uso de recursos através dos pedidos.
  - Incluindo links HATEOAS
    - links devem levar a outros recursos, que de alguma forma estão relacionados ao recurso atual.
      - Por exemplo, paginação em nossos recursos, podemos modelar a URL de paginação da seguinte forma:

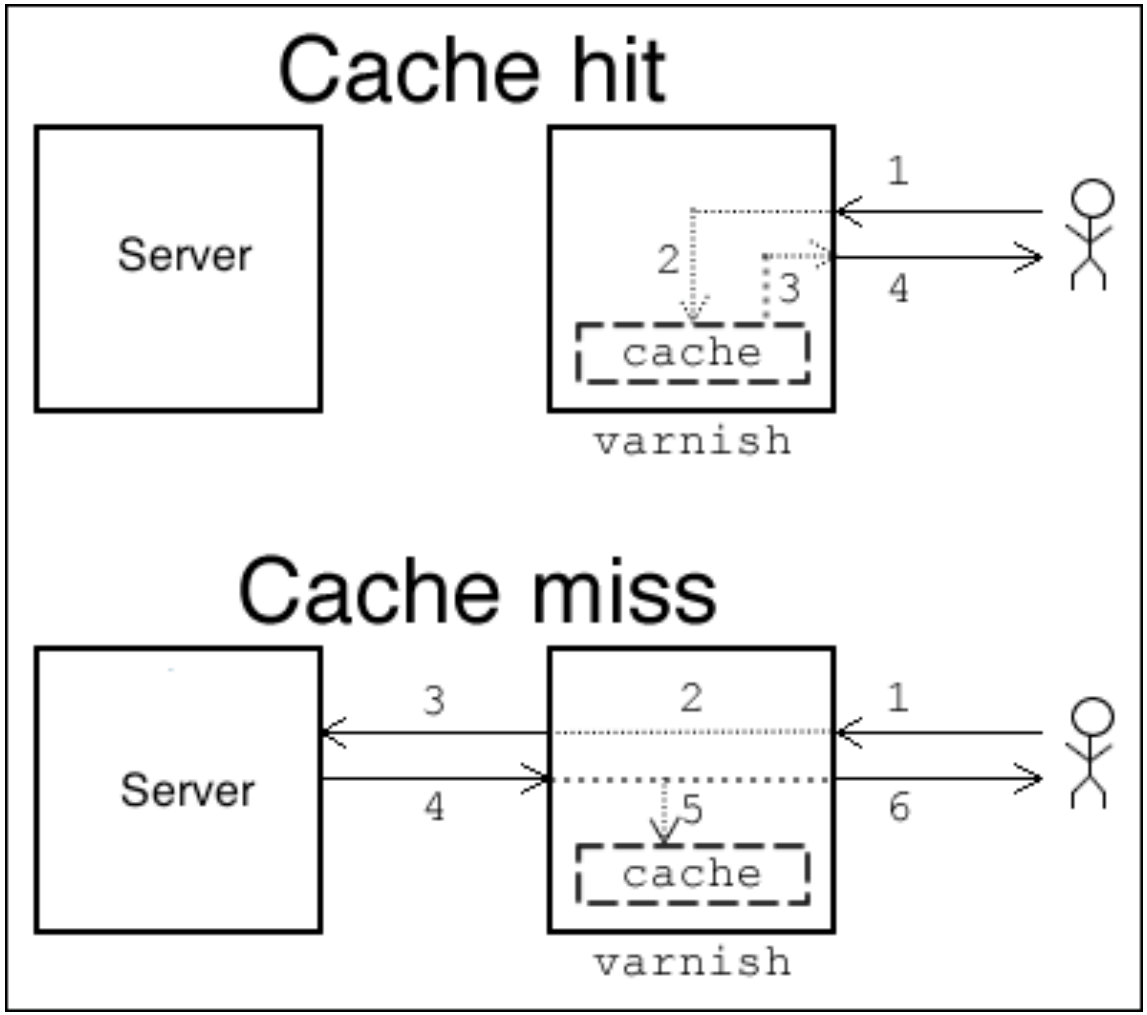
REST - CACHE

- Um dos grandes benefícios é alcançado com o uso das técnicas de cache
  - impacto direto sobre a escalabilidade da arquitetura.

- Exemplo: Quando é feita uma solicitação para recuperar um conjunto de dados, se este não mudar em certo período de tempo, usar o cache no lado do servidor traz o benefício de não ser necessário acessar o banco de dados para cada requisição do cliente, otimizando o sistema.

- Cache local, Cache-Control header
- Squid
- Varnish, Akamai

REST - CACHE


Cache hit
Server
1
2
3
4
cache
varnish
Cache miss
Server
3
2
1
4
5
6
cache
varnish

REST - Sistemas em Camadas

- Com o intuito de aperfeiçoar o requisito de escalabilidade da Internet, foi adicionado ao estilo REST a característica de divisão em camadas.

- Sistemas multicamada utilizam camadas para separar diferentes unidades de funcionalidade.

  - Encapsula complexidade
  - Simplicidade

REST - Code-on-Demand

- Estilo de code-on-demand
  - REST permite funcionalidade do cliente em baixar e executar código na forma de applets ou scripts.
  - Isto simplifica os clientes ao reduzir o número de recursos requeridos para ser pré-aplicado.
  - No entanto, reduz visibilidade e, portanto, é apenas uma restrição opcional dentro REST.

REST - Interface Uniforme

- A característica central de REST é sua ênfase em uma interface uniforme entre os componentes.

- REST define quatro requisitos de interface:
  - Identificação de recursos;
    - cada recurso tem um endereço global que pode ser usado para descoberta de serviços
  - Manipulação de recursos através de representações
    - Vários formatos
  - Mensagens auto-descritivas
  - Hipermídia como mecanismo de estado da aplicação
    - guiar o cliente fornecendo os próximos passos que este poderá seguir usando links
  - Estado da Aplicação e Estado do Recurso

REST

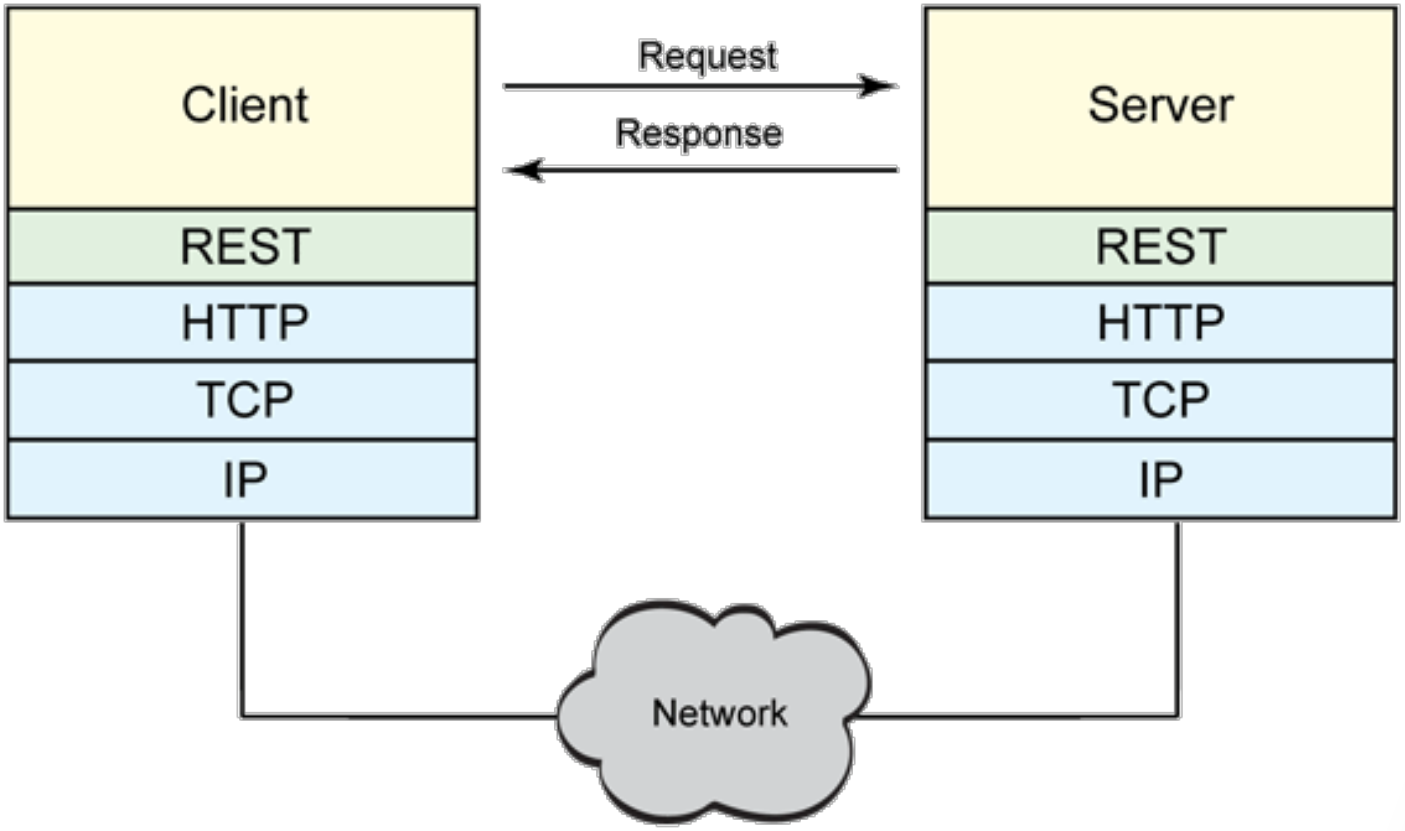
- A definição tradicional de REST deixa um monte de espaço aberto
  - REST é muito geral e nada sobre REST depende da mecânica de HTTP ou a estrutura de URIs.
  - Profissionais projetam os seus serviços, de acordo com seus próprios entendimentos de REST
    - maioria do serviços web, que se dizem REST, são na verdade híbridos de REST com RPC
  - Low REST
    - REST-RPC
    - Utiliza o HTTP: principalmente o métodos GET e POST para toda a API e a ação ser tomada é descrita na própria URI.
      - Muito comum por ser fácil de implementar.
  - High REST
    - RESTful
      - Utiliza HTTP e os quatros verbos (GET, PUT, POST e DELETE)
      - O uso correto ou não dos verbos e das URIs determina se uma aplicação é considerada RESTful ou não.
      - Web services que seguem o estilo de arquitetura REST.

RESTful

- Esta seção apóia-se principalmente nos conceitos apresentados por Richardson e Ruby (2007).
  - que cunharam o termo Serviços RESTful para designar serviços Web que seguem os critérios defendidos pelo paradigma REST.

- Adicionalmente, estes autores conectaram REST ao protocolo HTTP, trazendo um aspecto tecnológico concreto àquele grupo de critérios.
  - RESTful depende da mecânica de HTTP e da estrutura de URIs

Arquitetura



RESTful

- A abordagem RESTful é composta por basicamente cinco conceitos
  - Recurso, representação, identificador uniforme, interface unificada e escopo de execução
  - Além de três princípios (endereçabilidade, stateless e conectividade).
- Como todo serviço Web, um serviço RESTful recebe uma requisição discriminando o processamento a ser executado, e retorna uma resposta, detalhando o resultado obtido.

RESTful

- Recursos
  - Qualquer informação que puder ser nomeada pode ser considerada um recurso.
    - Tudo que é importante ao ponto de ser exposto pelo sistema na web.
    - Exemplos: usuário, produto, documento, imagem, outros recursos, etc.
  - São os substantivos
  - Todo recurso tem uma URI

RESTful

Nº	Serviços	Descrição
01	Consultar Empresa.	Permite a uma empresa obter a referência a outra.
02	Buscar preço e tempo estimado de um frete.	Informando a empresa destino, o peso da mercadoria e a data de envio, a empresa origem poderá obter o preço e o tempo estimado deste frete.
03	Agendar um frete	Permite a empresa origem agendar um frete, será necessário informar a empresa destino, a data de envio e o peso da mercadoria.
04	Editar um frete	Permite que um frete seja alterado ou excluído.
05	Cancelar um frete	Permite que um frete seja excluído.

RESTful

Recursos Pré-definidos	
Empresas	Recurso único que agrupará recursos do tipo empresa.
Fretes	Recurso único que agrupará recursos do tipo frete e funcionalidades não vinculadas a um frete existente.
Recursos que representam dados	
Empresa	Existirá um recurso deste para cada instância de empresa.
Frete	Existirá um recurso deste para cada instância de frete.
Recursos que representam funcionalidades	
Cotação	Recurso que representa a funcionalidade de calcular o preço e o tempo estimado (em dias corridos) de um frete.

RESTful

- Os recursos são coisas e não ações, por isso os nomes utilizados na URI devem ser padronizados como substantivos.
  - Por exemplo, para um recurso usuário faça com que o @GET seja executado caso seja feita uma solicitação GET à URI /user/{id}.
  - Como há uma padronização neste sentido, é um anti-padrão mapear a URI como /user/getUser/{id}
    - Pois as operações básicas – adicionar, recuperar, atualizar e remover – serão definidas pelos métodos HTTP.

RESTful

- Identificador Uniforme
  - Cada recurso está necessariamente associado a pelo menos um URI (Uniform Resource Identifier) que atua simultaneamente como nome e localizador deste recurso.
    - URIs compõe um namespace global,e utilizando URIs para identificar seus recursos chave significa ter um ID único e global.
  - Caso um determinado objeto não tenha um URI associado, não poderá ser considerado um recurso.
    - Entretanto, um único recurso pode ser referenciado por um número ilimitado de URIs.

RESTful

Nº	Padrão da URI	Objetivos
1	➢ /empresas	• Lista recursos do tipo empresa.
1.1	→ /{id}	• Representa um recurso do tipo empresa já cadastrado.
2	➢ /fretes	• Lista recursos do tipo frete. • Recurso-fábrica[4] para recursos do tipo frete.
2.1	→ /{id}	• Representa um recurso do tipo frete já cadastrado.
2.2	→ /cotacao	• Informa o preço e o tempo estimado de um frete.

RESTful

- Representação
  - Conceitualmente, uma representação é qualquer informação útil sobre o estado do recurso.
  - Tecnicamente, uma representação é uma serialização do recurso na sintaxe escolhida
    - Alguns formatos comumente utilizados são: XML, XHTML (Extensible Hypertext Markup Language), Imagens (PNG, JPEG,…), JSON (JavaScript Object Notation), Texto Puro, RDF (Resource Description Framework) e etc…

RESTful

- Representação

Accept: text/xml,

```
<usuario>
  <login>blpsilva</login>
</usuario>
<usuario>

    <login>teste999</lo
    gin>
</usuario>
```

OU

application/json

```
"usuarios":[
  {
    "login":"blpsilva",
  },
  {
    "login":"teste999",
  }
]
```

RESTful

- Interface Unificada
  - Segundo o paradigma RESTful, a ação a ser executada é definida diretamente pelo método HTTP.
  - O protocolo oferece cinco métodos principais: GET, HEAD, POST, PUT e DELETE. Todos os métodos são aplicados nos recursos, ou seja, nos objetos gerenciados pelo serviço.
  - Mais especificamente, um método HTTP é executado para um determinado URI.
  - Apesar da simplicidade, esta característica é extremamente poderosa, pois define uma interface unificada para todos os serviços.

RESTful - VERBOS

Operação CRUD	Método HTTP	Definição
Consultar	*Get*	Recupera a representação de um Recurso.
Excluir	*Delete*	Remove um Recurso.
Incluir	*Put*	Cria um recurso, a URL é definida pelo cliente.
	Post	Cria um recurso subordinado. A URL é definida pelo servidor e o método é utilizado no recurso superior ao qual será criado.
Alterar	*Put*	Sobrescreve um recurso.
	Post	Altera o recurso, sem sobrescrever os dados já existentes do recurso.

Tabela 01 (Richardson; Ruby, 2007)

RESTful - VERBOS

Nº	Padrão da URI	Métodos	Operação CRUD	Parâmetros de consulta
1	➢ /empresas	*Get*	Consultar	nome, cidade
1.1	→ /{id}	*Get*	Consultar	-
2	➢ /fretes	*Get*	Consultar	origem, destino, dataenvio, peso, dataenvioinicio, dataenviofim
		Post	Incluir (um Frete)	-
2.1	→ /{id}	*Get*	Consultar	-
		Delete	Excluir	-
		Put	Alterar	-
2.2	→ /cotacao	*Get*	Consultar	origem, destino, dataenvio, peso

RESTful - GET

- Os métodos Get
- Usado para solicitar uma representação do recurso especificado.
- Para os exemplos, imaginaremos que temos um recurso user implantado em http://www.exemplo.com/recursos/user/.
  - Na prática, fazendo uma requisição GET em

    http://www.exemplo.com/recursos/user/10

  - Será retornado o usuário cujo id é igual a 10. Caso nosso alvo fosse http://www.exemplo.com/recursos/user, teríamos a lista de todos os usuários.

RESTful - PUT

- Atualiza a representação de um recurso.
- Os dados devem ser enviados no corpo da requisição (body).
  - Além disso, se a URI do pedido não apontar para um recurso existente, e é permitido pela lógica do serviço que a URI possa ser definida como um novo recurso, o servidor de origem pode criar o recurso com esta URI.
  - No nosso exemplo, se quisermos atualizar o password do usuário com id “10”, bastaria fazer uma solicitação PUT para www.exemplo.com/recursos/user/10 e passar um JSON pelo body contendo os novos dados.

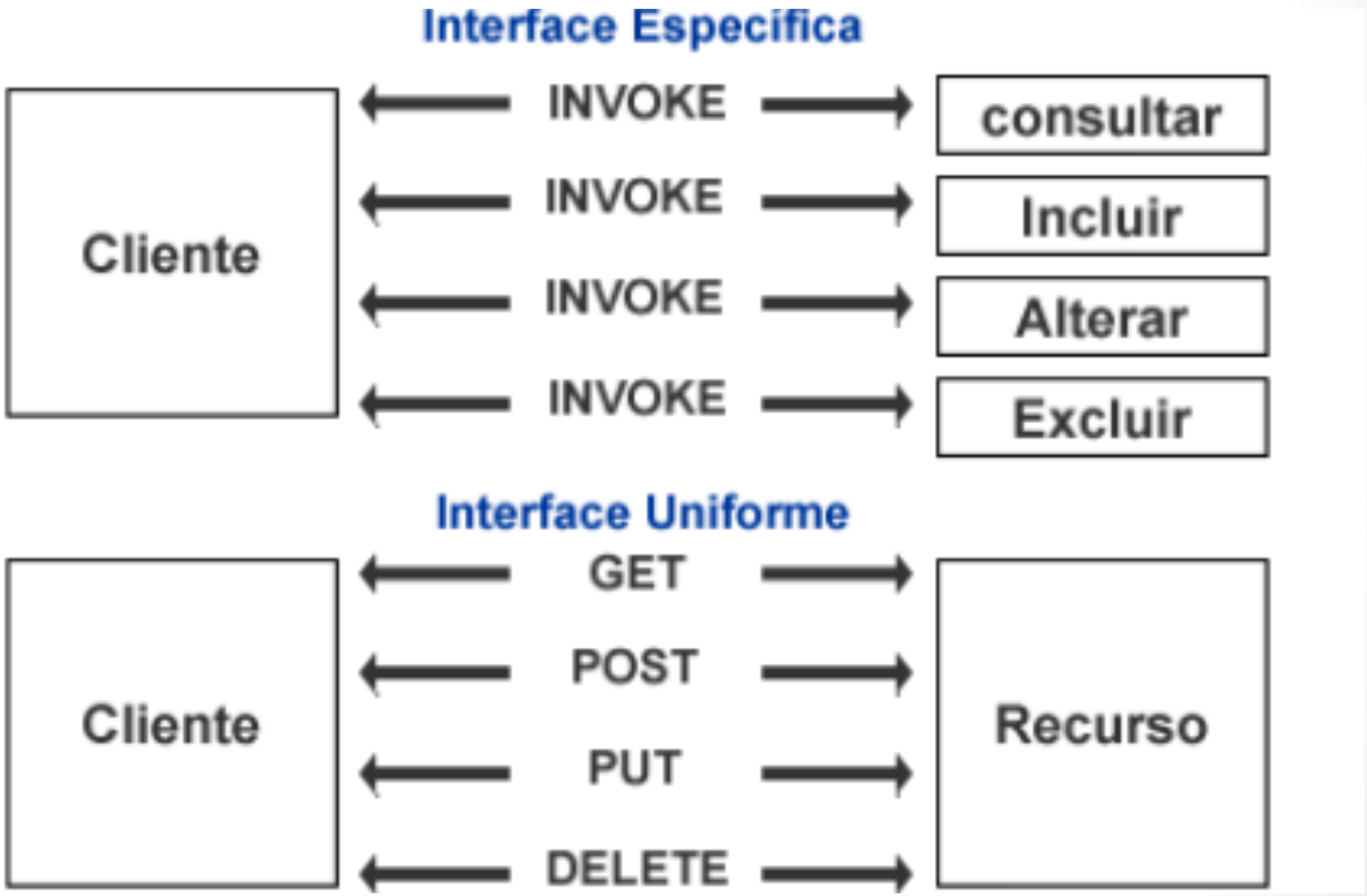
RESTful - DELETE

- Deleta um recurso específico.
  - Não tem corpo (body).
  - Se requisitarmos um DELETE à URI http://www.exemplo.com/recursos/user/10, significa que queremos que o servidor remova o usuário cujo id é igual a 10.

RESTful - POST

- O método Post
  - Submete os dados a serem processados para um recurso alvo, o que pode resultar na criação de um novo e/ou nas atualizações dos recursos existentes.
  - Os dados são incluídos no corpo do pedido (body).
- Possui diversas funções:
  - Cria um novo recurso subordinado: Recursos que não podem ter a sua URI definida pelo cliente.
    - Como exemplo desta situação, podem-se citar recursos que ao serem criados são persistidos em um banco de dados, e em sua URI seja informado o registro deste recurso.
  - Altera um recurso alterando seu estado sem sobrescrevê-lo

RESTful

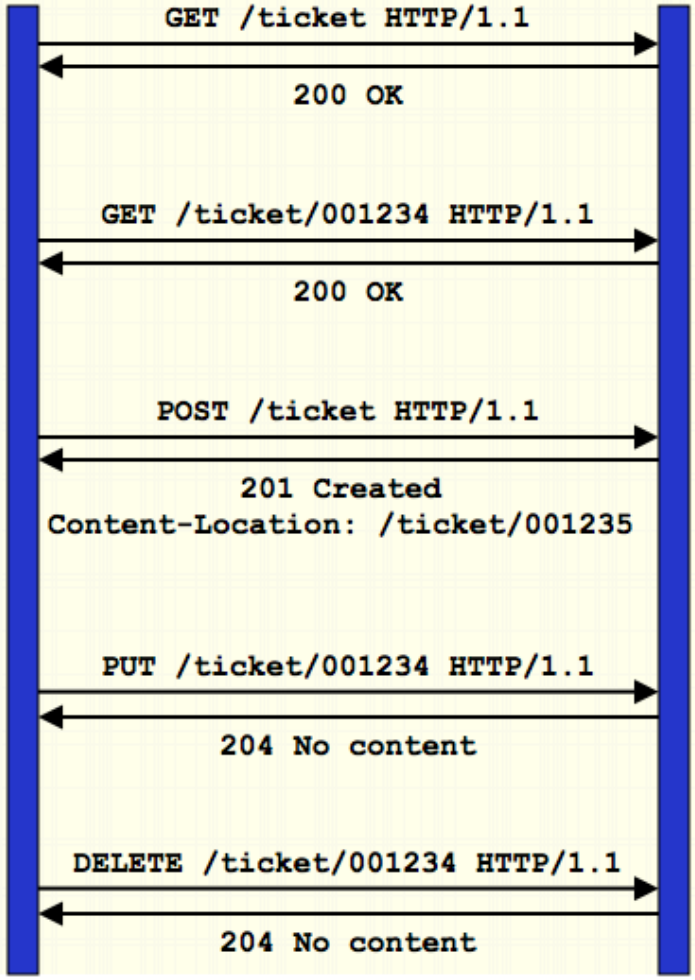

Interface Especifica
Cliente
INVOKE
consultar
INVOKE
Incluir
INVOKE
Alterar
INVOKE
Excluir
Interface Uniforme
Cliente
GET
POST
PUT
DELETE
Recurso

RESTful

Não RESTful		
Verbo	**URI (substantivo)**	**Ação**
POST	/bookmarks/create	Criar
GET	/bookmarks/show/1	Visualizar
POST	/bookmarks/update/1	Alterar
GET/POST	/bookmarks/delete/1	Apagar

RESTful		
Verbo	**URI (substantivo)**	**Ação**
POST	/bookmarks	Criar
GET	/bookmarks/1	Visualizar
PUT	/bookmarks/1	Alterar
DELETE	/bookmarks/1	Apagar

RESTful



Retrieve a list of all current tickets

Retrieve a particular ticket

Create a new ticket. Retrieve new ticket URI from HTTP Headers.

Update a ticket.

Delete a ticket.

RESTful

- Cada requisição gera uma resposta do servidor. Esta resposta vem codificada em elementos chamados status-code seguindo a regra:

Status	Utilização
200	Solicitação realizada com sucesso
201	Recurso criado com sucesso
204	Sucesso na solicitação. Resposta sem corpo.
400	Requisição inválida
404	Recurso inexistente
405	Método HTTP não permitido para a URI
415	Tipo de conteúdo não suportado
500	Erro interno no servidor

RESTful

- Além do REST, é possível construir web services usando SOAP. Veremos a seguir algumas situações onde cada estilo é mais recomendado.

- É interessante nas seguintes situações:

  - Os serviços web são totalmente sem estado;

  - O armazenamento em cache pode ser aproveitado para o desempenho;

  - O formato dos dados não deve obrigatoriamente ser XML

RESTful

- Quando não é interessante utilizar serviços RESTful
  - Um contrato formal deve ser estabelecido entre o cliente e o serviço
  - Devem ser atendidos requisitos não-funcionais como transações e autenticação
  - É necessário estabelecer uma sessão entre o cliente e o serviço (serviços statefulls)

RESTful e SOAP

Fator de Comparação	***Web services* WS-***	***Web services* REST**
Tamanho de Pacote HTTP	584 bytes	399 bytes
Armazenamento de estados	Permite	*Stateless*
Abordagem de design	Baseado em operações ou verbos	Baseado em recursos ou substantivos (URIs)
Interface	Interface Não Uniforme	Interface Uniforme – GET, POST, UPDATE, DELETE.
Code on demand (COD)	Não é voltado a COD	Permite que código originado da requisição ao Servidor seja executado no lado Cliente.
Representação de dados	XML	Aberto
Descritor	WSDL	WSDL/WADL/Nenhum

GRANULARIDADE em RESTful

- Granularidade dividindo-a em duas grandes medidas:
  - granularidade grossa, tecnicamente chamada de coarse-grained
  - granularidade fina, tecnicamente chamada de fine-grained

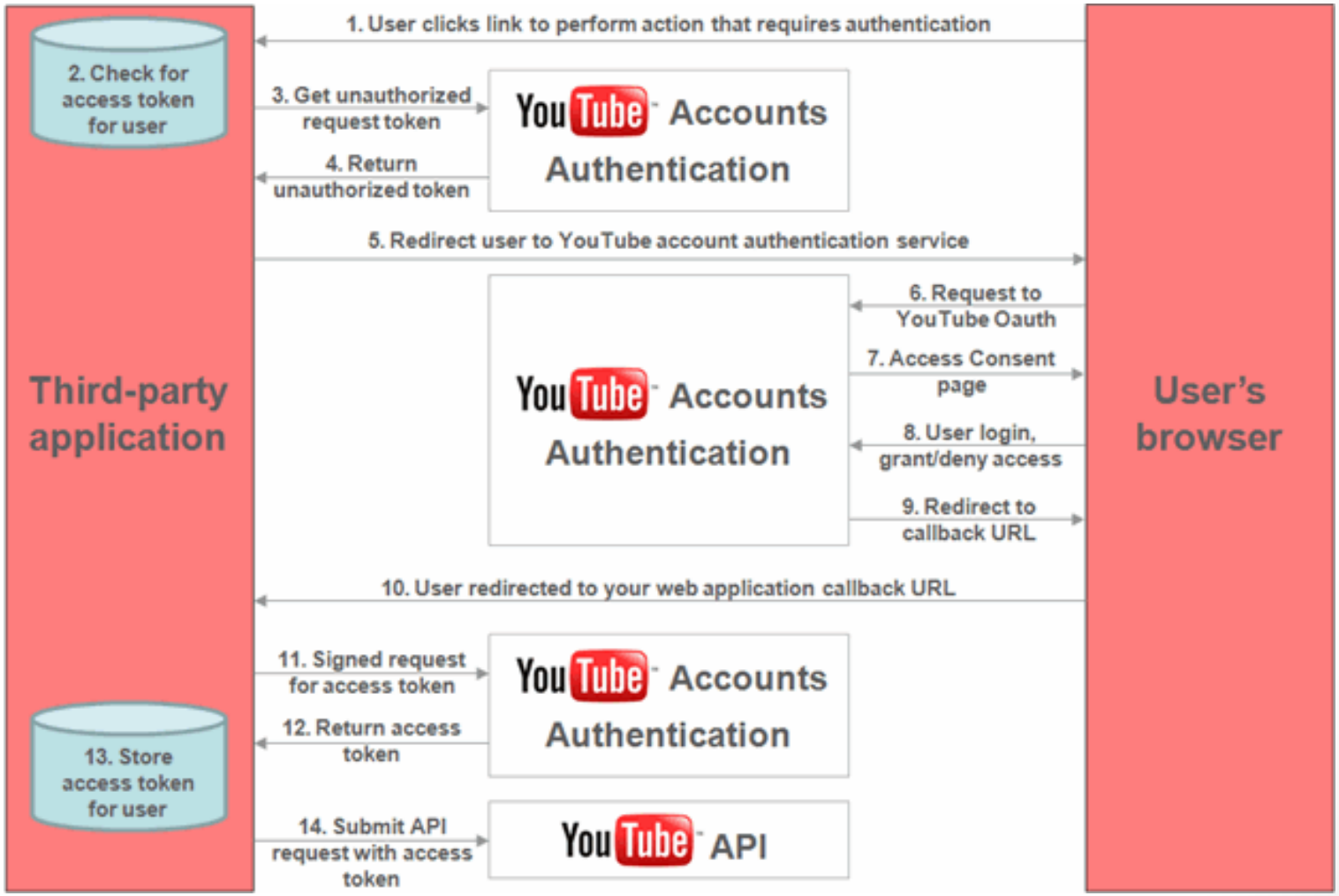
  - Em REST a granularidade pode ser fina (e focada nos dados).
    - Criação de serviços com interfaces CRUD (Create, Read, Update e Delete). Manipulação de registros dentro de uma determinada base de dados.

  - Nestes casos, o serviço será apenas uma espécie de wrapper para os dados, não fazendo nada além do que as operações básicas que todo banco de dados possui (INSERT, SELECT, UPDATE e DELETE).

SEGURANÇA

- Open Auth
- Versão atual 2.0
- Utilize Oauth para acessar/compartilhar seus dados protegidos.
- A aplicação precisa obter o token de acesso do provedor de serviços.
- O provedor de serviços utilizará o token de acesso após a autorização do usuário

SEGURANÇA


1. User clicks link to perform action that requires authentication
2. Check for access token for user
3. Get unauthorized request token
4. Return unauthorized token
YouTube Accounts Authentication
5. Redirect user to YouTube account authentication service
6. Request to YouTube Oauth
7. Access Consent page
8. User login, grant/deny access
9. Redirect to callback URL
YouTube Accounts Authentication
10. User redirected to your web application callback URL
11. Signed request for access token
12. Return access token
YouTube Accounts Authentication
13. Store access token for user
14. Submit API request with access token
YouTube API
Third-party application
User's browser

SEGURANÇA

- Certificação
  - Todas as chamadas RESTful devem ser executadas sobre HTTPS, com um certificado assinado por uma autoridade certificadora confiável.
    - Todos os clientes devem validar o certificado antes de interagir com o servidor.
  - Enviar credenciais dentro do header do HTTPS, fazendo autenticação a cada chamada.

Exemplos

- Facebook
  - https://developers.facebook.com/docs/graph-api
- Twitter
  - https://dev.twitter.com/docs/api
- Google Maps
  - https://developers.google.com/maps/documentation/webservices/?hl=pt-br