REST

Eduardo Ferreira dos Santos

Ciência da Computação
Centro Universitário de Brasília – UniCEUB

Outubro, 2016

Sumário

Recapitulando

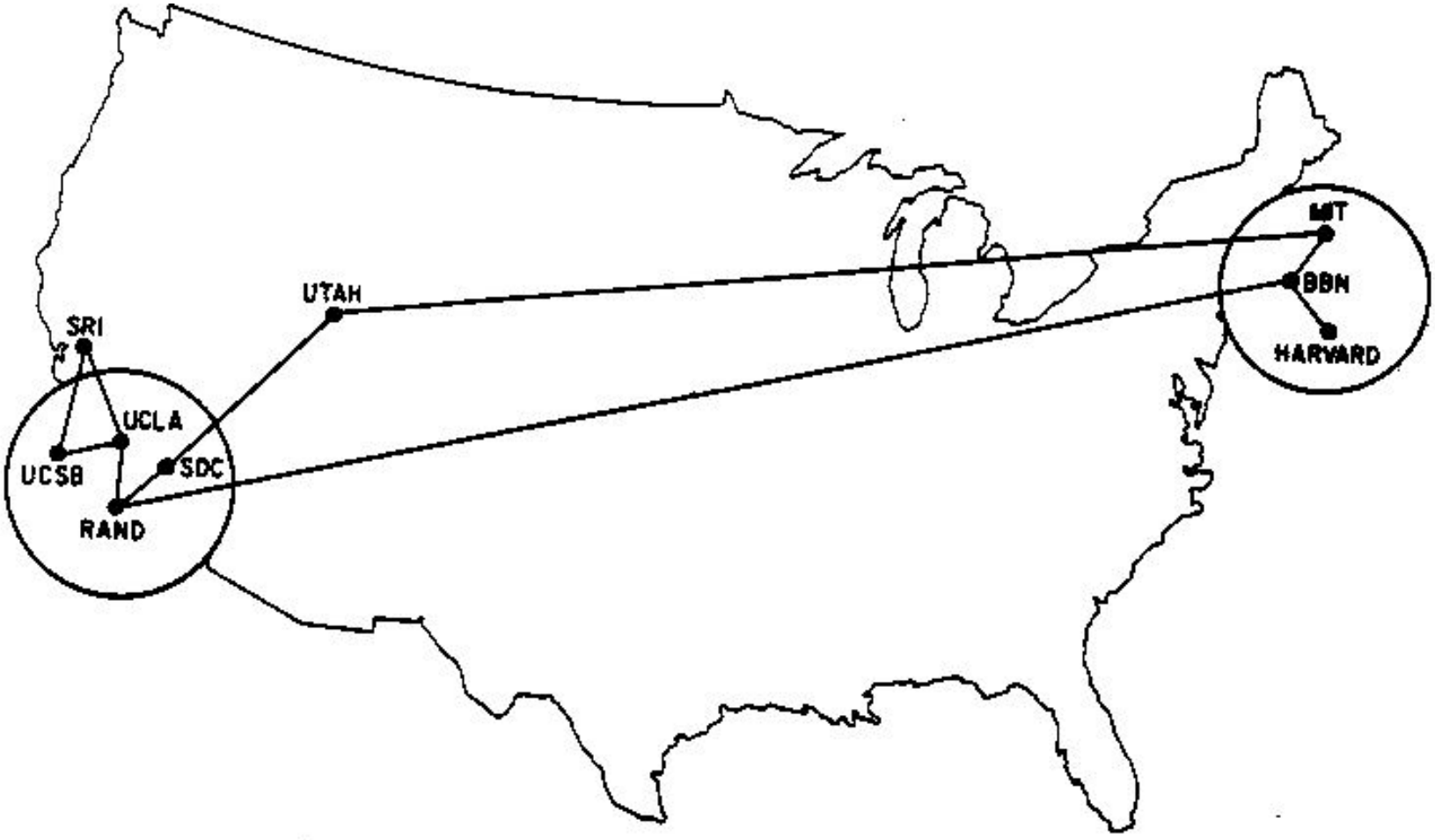

Figura 1.1: Recordando a razão da existência da Internet [Heart et al., 1978]

Compartilhamento de recursos

- Os computadores foram construídos primeiramente para compartilhar recursos;
- Importância do hardware e vendor lock-in;
- Como fazer o compartilhamento de recursos em uma arquitetura distribuída;
- Conceito de Web Services.

Sistemas distribuídos

Um sistema distribuído consiste em um conjunto discreto de agentes de software que devem trabalhar em conjunto para executar uma tarefa. Além disso, os agentes em um sistema distribuído não operam no mesmo ambiente de processamento, de forma que eles precisam se comunicar através de pilhas de protocolo de hardware e/ou software em uma rede.
[Booth et al., 2004]

- A comunicação é menos confiável e mais lenta;
- Os programadores precisam se preocupar com questões como:
  - Latência;
  - Concorrência;
  - Possibilidade de falha.

Definição

Um Web Service é um sistema de software desenhado para suportar a interoperabilidade entre máquinas em uma rede. Possui uma interface descrita em um formato processado por máquina (normalmente WSDL). Outros sistemas interagem com o Web service utilizado as operações contidas em sua descrição utilizando mensagens SOAP, normalmente convertidas em HTTP utilizando serialização XML, junto com outros padrões da Web relacionados. [Booth et al., 2004].

Arquitetura

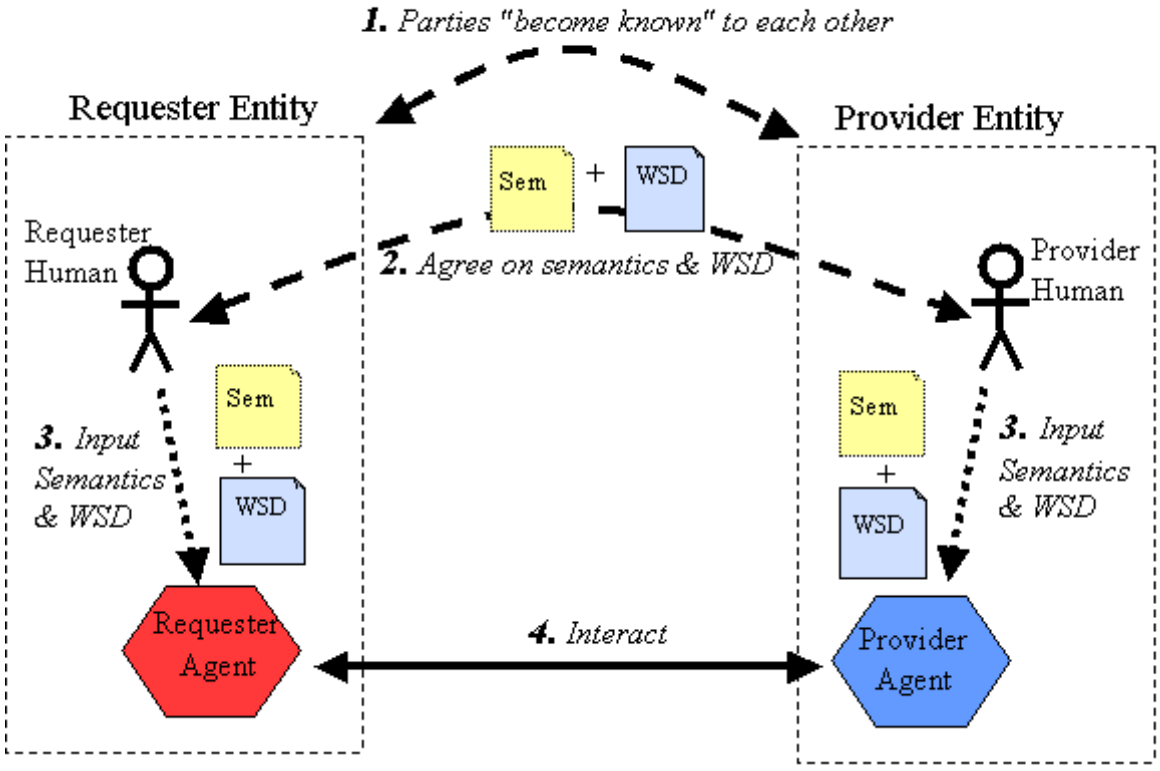

Figura 1.2: Processo de conexão a um Web service [Booth et al., 2004]

SOA

- *Service Oriented Architecture* – SOA – é um tipo de sistema distribuído que se caacteriza por algumas propriedades [Booth et al., 2004]:

  Visão Lógica Se preocupa com o que o sistema faz, e não como faz;

  Orientado a mensagens O serviço é definido em termos das mensagens trocadas entre os sistemas;

  Orientado à descrição Utilização de metadados processáveis por máquina;

  Granularidade Número pequeno de operações com mensagens grandes e complexas;

  Orientados à rede Normalmente são programados para funcionar em rede;

  Neutralidade de plataforma As mensagens devem ser enviadas em um formato independente de plataforma, normalmente utilizando XML.

Camadas

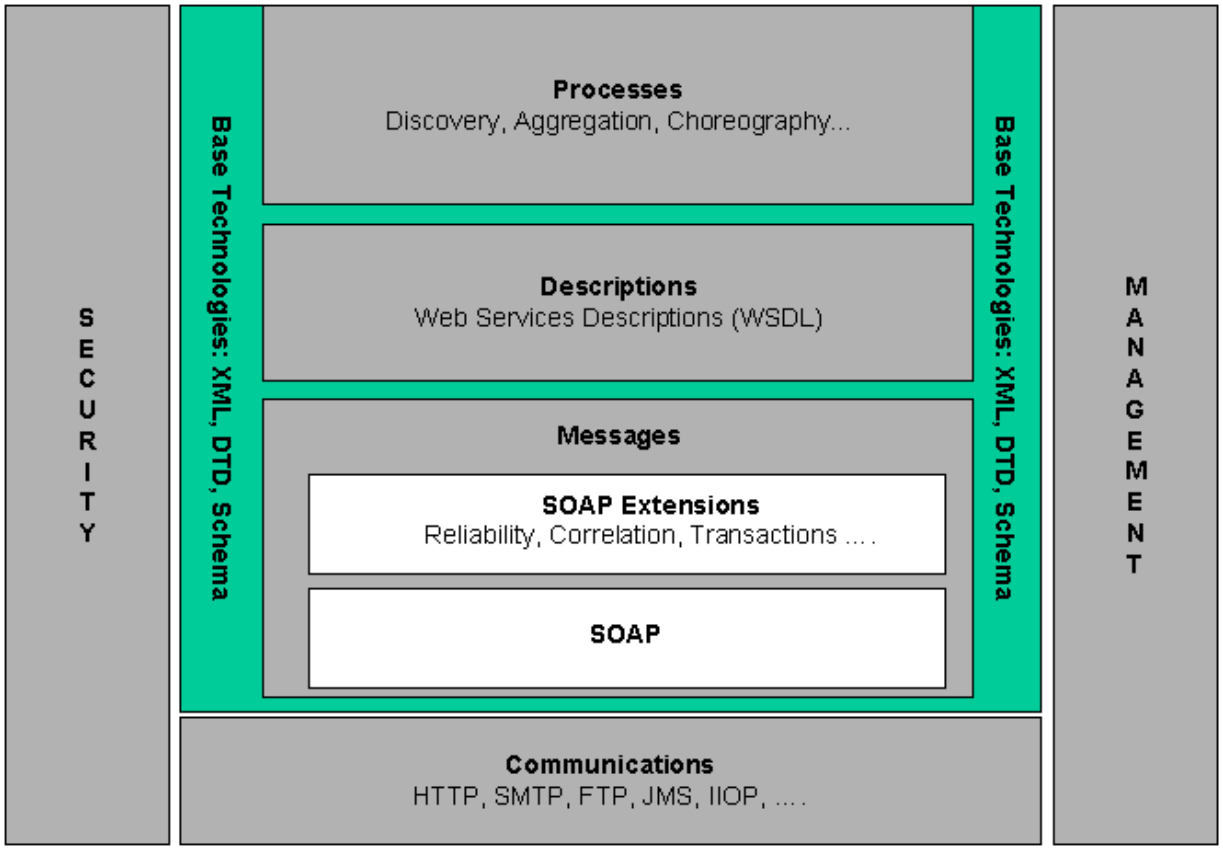

Figura 1.3: Camadas em um Web service [Booth et al., 2004]

Exemplo de SOA

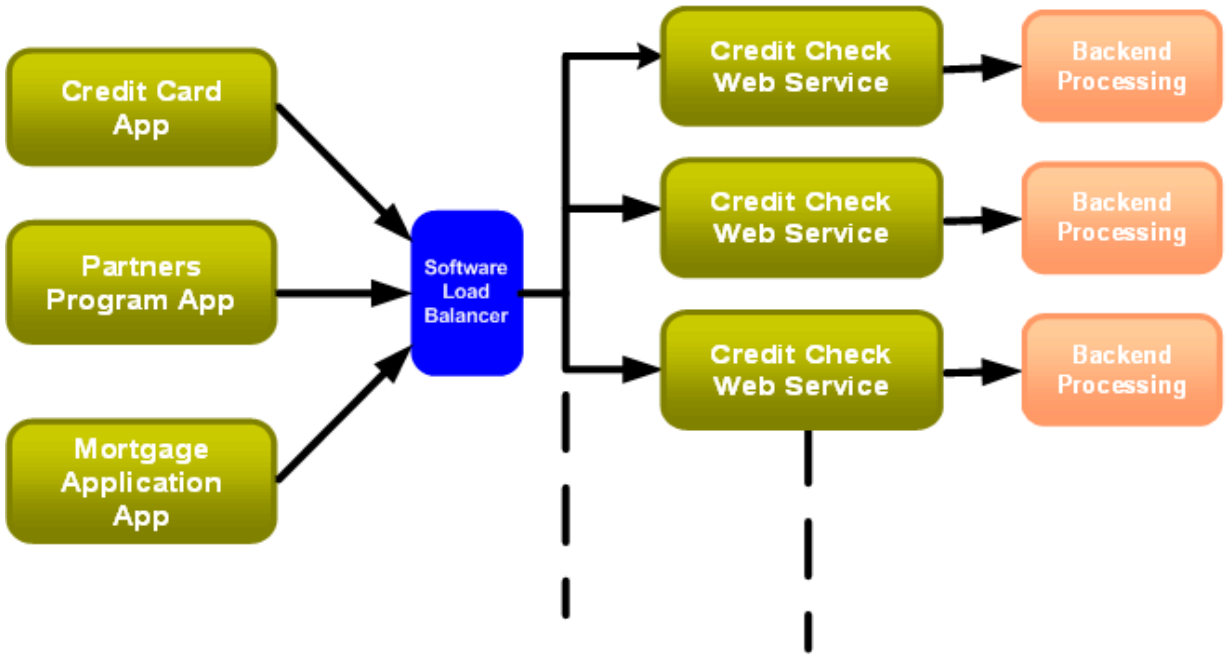

Figura 1.4: Exemplo de aplicação SOA [1]

[1] Fonte: http://docs.oracle.com/cd/E13156_01/wloc/docs103/example/ex_service.html

Topologia SOA

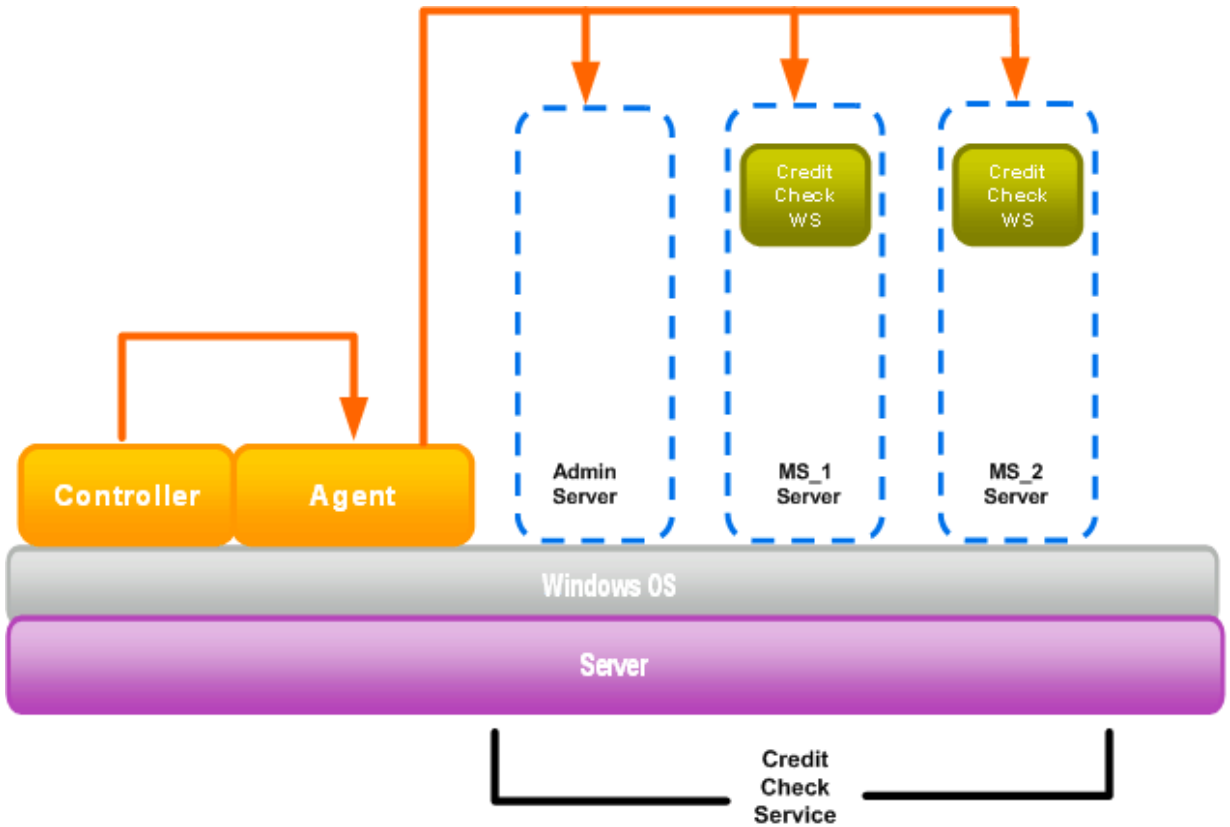

Figura 1.5: Exemplo de topologia aplicação SOA [2]

[2] Fonte: http://docs.oracle.com/cd/E13156_01/wloc/docs103/example/ex_service.html

1 Web Services

2 REST
- Arquitetura
- Implementação

Definições [Elkenstein, 2008]

- REST – *Representational State Transfer*;
- Se baseia em um protocolo de comunicações stateless, cliente-servidor, passível de cache;
- O REST é um estilo arquitetural para construir aplicações baseadas na rede;
- Em praticamente todos os casos, o protocolo HTTP é utilizado;
- Ao invés de utilizar chamadas complexas como RPC, CORBA ou SOAP, o protocolo HTTP é utilizada para realizar chamadas entre as máquinas;
- É possível imaginar a *World Wide Web* – WWW – como uma implementação baseada na arquitetura REST.

Características

- Os agentes identificam os objetos no sistema através de URI – *Uniform Resource Identifier*;
- A comunicação de estado, representação e descrição dos objetos é realizada através de um dos formatos já utilizados na Web (XML, HTML, PNG, CSS, JPEG);
- A troca de informações é realizada através de um dos protocolos que utilizam URI para identificar os objetos direta ou indiretamente;
- Utilização das operações de CRUD associadas a métodos;
- Apesar de ser bastante simples, não há nada que seja definido para *Web Services* que não possa ser implementado utilizando REST;
- O REST não é um padrão. Cada linguagem/ferramenta pode possuir sua própria implementação.

Interface Uniforme

- Principal característica do REST: interface uniforme entre os componentes [Fielding, 2000];
- Princípio da neutralidade em Engenharia de Software:
  - A arquitetura geral do sistema é simplificada;
  - A visibilidade das das interações é melhorada;
  - As implementações são desacopladas dos serviços que fornecem.
- Ao exigir um formato padrão para circulação de informações, assume alguma degradação de performance;
- A interface REST é otimizada para funcionar com o protocolo de transmissão HTTP.

Restrições

- Restrições arquiteturais necessárias para obter uma interface uniforme, guiando o comportamento dos componentes [Fielding, 2000]:
  1. Identificação de recursos;
  2. Manipulação dos recursos através de representações;
  3. Mensagens auto-descritivas;
  4. Utilização de hipermídia como máquina de estados para as aplicações.
- As quatro restrições formam a definição de REST;
- Cada operação no REST deve ser auto-contida, ou seja, cada requisição contém todas as informações necessárias ao servidor para realizar a transação.

Código on-demand

- O conjunto de funcionalidades dos clientes pode ser estendido baixando e executando código no formato de applets ou scripts;
- Aumenta a extensibilidade do sistema;
- Nem todas as *features* precisam estar disponíveis no momento da primeira implementação.

Arquitetura e restrições

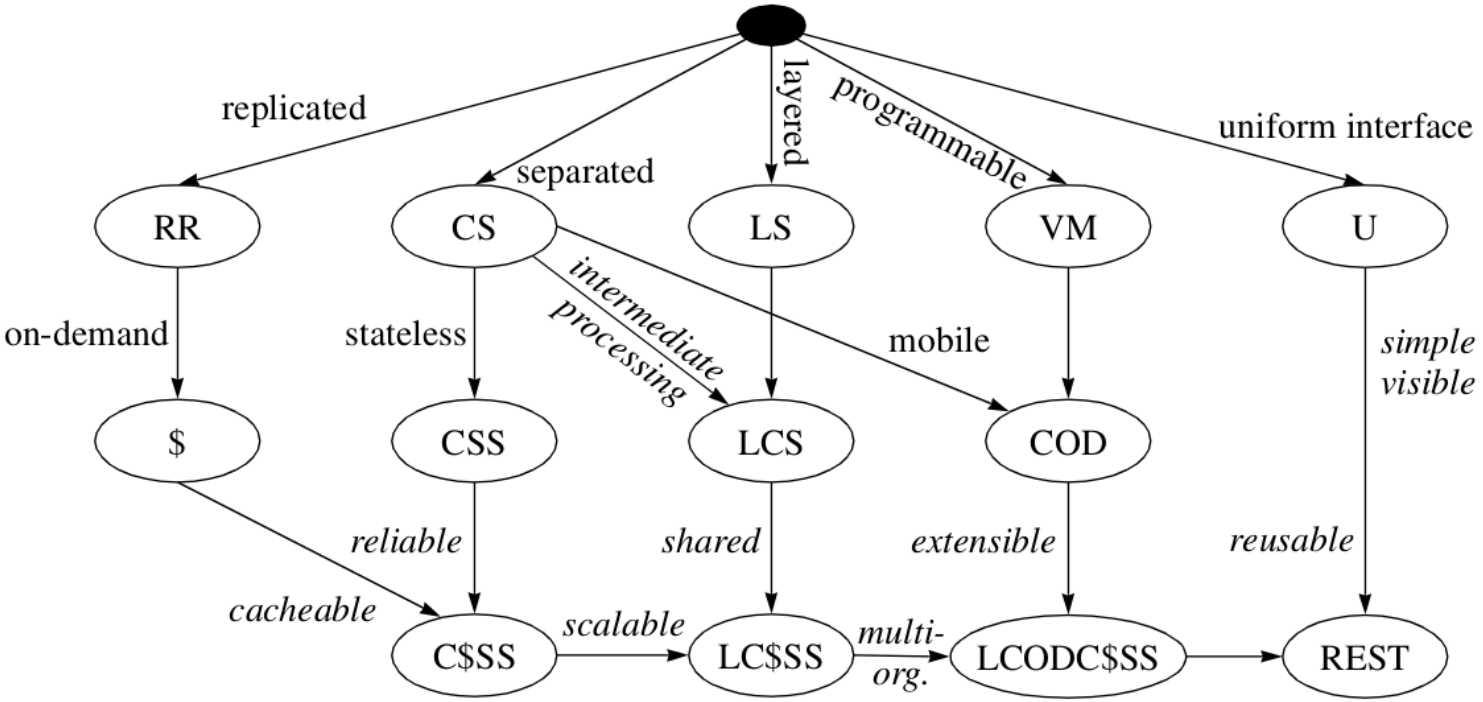

Figura 2.1: Derivação da arquitetura do REST com base nas restrições [Fielding, 2000]

Componentes arquiteturais

- Componentes arquiteturais chave para uma implementação válida de REST [Elkenstein, 2008]:

  Recursos Identificados logicamente por URL. Tanto o estado como a funcionalidade são representados através dos recursos (*resources*);
  - Os recursos são universalmente endereçáveis utilizando os conceitos de URI e URL;
  - Tratam-se de elementos chave do desenho de um sistema RESTful;
  - Os recursos não são vistos como objetos que recebem métodos, e sim como elementos auto-contidos, que devem conter todas as informações necessárias para sua representação.

  Dados linkados Os recursos contém toda informação relevante à sua descrição ou contém hyperlinks para outros recursos;

  Stateless Toda requisição deve conter todos os dados necessário para completá-la, e não deve depender de interação prévia com outras requisições.

SOAP x REST

- Problema: encontrar um usuário numa agenda de telefones com base no ID [Fielding, 2000].
- Chamada SOAP:

Listing 1: Chamada SOAP para encontrar um usuário em um catálogo telefônico [Elkenstein, 2008]

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

- Chamada HTTP:

GET `http://www.acme.com/phonebook/UserDetails/12345` HTTP/1.1

Simplicidade

- Na chamada SOAP, todo o conteúdo da chamada deve ser embutido no corpo da requisição;
- O resultado provavelmente será um arquivo XML, embutido dentro do "envelope" SOAP;
- No caso da chamada REST, trata-se apenas de uma requisição GET sem nenhum dado;
- O resultado será escrito diretamente na conexão, sem a necessidade de estar dentro de nenhum "envelope";
- Ambas podem trazer o mesmo conjunto de informações, mas solicitar e tratar a resposta é mais simples no REST.

Utilização de parâmetros

- Vamos supor agora que vamos adicionar filtros à chamada REST;
- É possível adicionar vários parâmetros utilizando a especificação do protocolo HTTP:

```
GET http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe HTTP/1.1
```

- A resposta a uma requisição REST pode ser em qualquer formato, inclusive XML;
- Contudo, requisições REST raramente utilizam XML na resposta. Na maior parte dos casos, tratar o conteúdo do XML não é necessário para obter o resultado esperado pela requisição;

Respostas

- A requisição REST vai gerar uma resposta HTTP;
- Sempre é necessário validar a resposta. A validação deve ocorrer de duas formas:
  1. Código de retorno do protocolo HTTP. Ex.: HTTP 200 Ok
  2. Validade dos dados da requisição. Ex.: parsing de XML

Listing 2: Resposta de chamada XML [Elkenstein, 2008]

```
<parts-list>
 <part id="3322">
  <name>ACME Boomerang</name>
  <desc>
   Used by Coyote in <i>Zoom at the Top</i>, 1962
  </desc>
  <price currency="usd" quantity="1">17.32</price>
  <uri>http://www.acme.com/parts/3322</uri>
 </part>
 <part id="783">
  <name>ACME Dehydrated Boulders</name>
  <desc>
   Used by Coyote in <i>Scrambled Aches</i>, 1957
  </desc>
  <price currency="usd" quantity="pack">19.95</price>
  <uri>http://www.acme.com/parts/783</uri>
 </part>
</parts-list>
```

JSON

- Mesma resposta no formato JSON:

Listing 3: Resposta de chamada no JSON

```
{
    [
        {
            "id": 3322,
            "name": "ACME Boomerang",
            "desc": "Used by Coyote in <i>Zoom at the Top</i>, 1962",
            "price": {
                    "currency": "usd",
                    "quantity": 1,
                    "value": 17.32
            },
            "uri": "http://www.acme.com/parts/3322"
        },
        {
            "id": 783,
            "name": "ACME Dehydrated Boulders",
            "desc": "Used by Coyote in <i>Scrambled Aches</i>, 1957",
            "price": {
                    "currency": "usd",
                    "quantity": "pack",
                    "value": 19.95
            },
            "uri": "http://www.acme.com/parts/783"
        }
    ]
}
```

Vantagens JSON

- Utilização de *Javascript Object Notation* – JSON;
- Transporte transparente de objetos utilizando HTTP;
- Maior facilidade de serialização em diferentes linguagens;
- Na maior parte das linguagens o formato JSON é nativo para objetos.

AJAX

- *Asynchronous Javascript and XML* – AJAX;
- Método de comunicação que torna o desenvolvimento mais interativo;
- Utilização de requisições do tipo `XMLHttpRequest`;
- O AJAX segue os princípios do REST: cada requisição do tipo `XMLHttpRequest` pode ser vista como uma requisição a um serviço do REST;
- Aproveita-se das chamadas nativas do *browser* do usuário para enviar comandos ao servidor.

Comandos

- Organização de comandos das requisições com base no método HTTP:

GET	Consultas em modo somente leitura. Similares ao comando SELECT;
POST	Inserção de dados. Similares ao comando INSERT;
PUT	Atualização de dados. Similares ao comando UPDATE;
DELETE	Remoção de dados. Similares ao comando DELETE.

- Cada comando representa uma operação a ser implementada na aplicação;
- Questões relativas à segurança e isolamento.

Dados do REST

Data Element	Modern Web Examples
resource	the intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation	HTML document, JPEG image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

Figura 2.2: Elementos de dados em aplicações REST [Fielding, 2000]

Aplicações RESTful

- Alguns princípios de desenvolvimento devem ser seguidos para tornar a aplicação RESTful, ou compatível com o REST [Elkenstein, 2008]:
  1. Utilize os elementos de dados (Figura 31);
  2. Não utilize URLs físicas. O recurso deve possuir um significado tangível. Ex.:

     Ruim `http://www.acme.com/inventory/product003.xml`
     Bom `http://www.acme.com/inventory/product003`
  3. Não retorne uma quantidade de informações demasiadamente grande. Sempre que possível, forneça paginação;
  4. Tente garantir que a reposta a uma requisição REST não seja alterada em demasia e forneça documentação;
  5. Forneça sempre que possível URL's reais para novas ações. Ex.: ao retornar uma lista de produtos, cada produto deve conter sua URL para a operação GET (ver exemplo 3)
  6. Utilize os comandos HTTP para produzir mudanças de estado nos objetos.

OBRIGADO!!!
PERGUNTAS???


Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (2004).
Web services architecture.
Technical report.
Disponível em `https://www.w3.org/TR/ws-arch/` Acessado em 06/10/2016.


Elkenstein, M. (2008).
Learn rest: A tutorial.
Disponível em `http://rest.elkstein.org/` Acessado em 20/10/2016.


Fielding, R. T. (2000).
Architectural styles and the design of network-based software architectures.
PhD thesis, University of California, Irvine.


Heart, F., McKenzie, A., McQuillian, J., and Walden, D. (1978).
Arpanet completion report.

Technical report.
Disponível em:
http://som.csudh.edu/fac/lpress/history/arpamaps/ Acessado em 25/07/2016.