Unichristus
Centro Universitário Christus

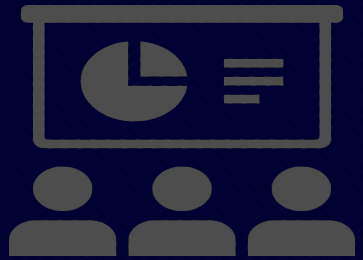
SISTEMAS DISTRIBUÍDOS

REST X SOAP

Lattes - linkedin
euristenhojr@gmail.com
http://www.unichristus.edu.br/

Euristenho Queiroz de Oliveira **Júnior**
Especialista em Engenharia de Software
MSc em Engenharia de Software

AGENDA



1. Apresentação
2. Livros
3. REST
4. REST x SOAP
5. ROA
6. Exercícios
7. Referências

FORMAÇÃO ACADÊMICA

- Graduado em Telemática/Telecomunicações - IFCE (2002 - 2008)
- Especialista em Engenharia de Software - FA7 (2011 - 2013)
- MSc em Engenharia de Software - UFPE (2011 - 2015)

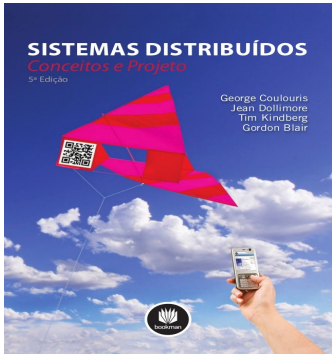
CURRÍCULO PROFISSIONAL

- Atuei 4 anos na empresa privada
- 9 anos no ambiente Público
- Atualmente Líder Técnico de 45 Projetos de Tecnologia na SEPOG/PMF

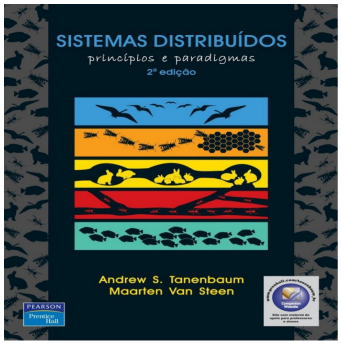
DOCÊNCIA

- Professor Substituto das Disciplinas de Sistemas de Informação – FA7 (2011 - 2012)
- Professor da Especialização em Sistemas WEB – FJN (2011 - 2012)
- Professor de Bancas de graduação em Sistemas de Informações – FA7 (2012)
- Professor dos Cursos de Tecnologia da Unifanor (2015 - ATUAL)
- Professor da Unichristus (2018 - ATUAL)

- **Sistemas Distribuídos - Conceitos e Projeto** - 5ª Ed. 2013 - George Coulouris, Tim Kindberg, Jean Dollimore



- **Sistemas Distribuídos, Princípios e Paradigmas** - 2ª Ed. 2007 - Andrew S. Tanembaum, Maarten Van Steen



OBJETIVOS

- Apresentar os conceitos básicos da computação distribuída e seus desafios como Heterogeneidade; Segurança; Tolerância a Falhas; Escalabilidade; Concorrência; Coordenação e Sincronização de processos; Comunicação interprocessos.
- Desenvolver competências e habilidades que auxiliem o profissional de Ciência da Computação a implementar os conceitos de sistemas distribuídos no desenvolvimento de sistemas de informação.
- Conhecer a aplicação desses conceitos em estudos de Casos que abordam arquiteturas e tecnologias modernas como RMI, CORBA e Web Services.

Agenda

1. **URI, URL e HTTP**
2. O que é SOAP
3. O que é REST?
4. Por que utilizar REST?
5. Restrições arquiteturais do REST
6. ‘Protocolo’ REST
7. REST x SOAP
8. Como organizar/gerenciar os recursos para web services REST
9. Arquitetura Orientada a Recursos (ROA – Resource Oriented Architecture)
10. Exemplos de serviços REST

Revisão

O que é uma URI e uma URL?

- URI – **Uniform Resource Identifier**
- URL – **Uniform Resource Locator**
- Localização ou endereço de um recurso na Web
  - Ex: http://sonybmg.com/cps/artists/shakira#bio
- Abordagem uniformizada para representar um recurso:
  - [scheme:][//authority][path][?query][#fragment]
- Recurso:
  - shakira
- Representação:
  - XML, HTML, RSS, JSON, ATOM …

Revisão

O que você lembra sobre o protocolo HTTP?

Revisão

- Definido na RFC 2616 (http://ietf.org/rfc/rfc2616.txt)
- HTTP é um protocolo sem estado (stateless)
- Métodos HTTP(em ordem de popularidade):
  - GET, POST
  - PUT, DELETE
  - HEAD, OPTIONS, TRACE

- Método HTTP x VERBOS:

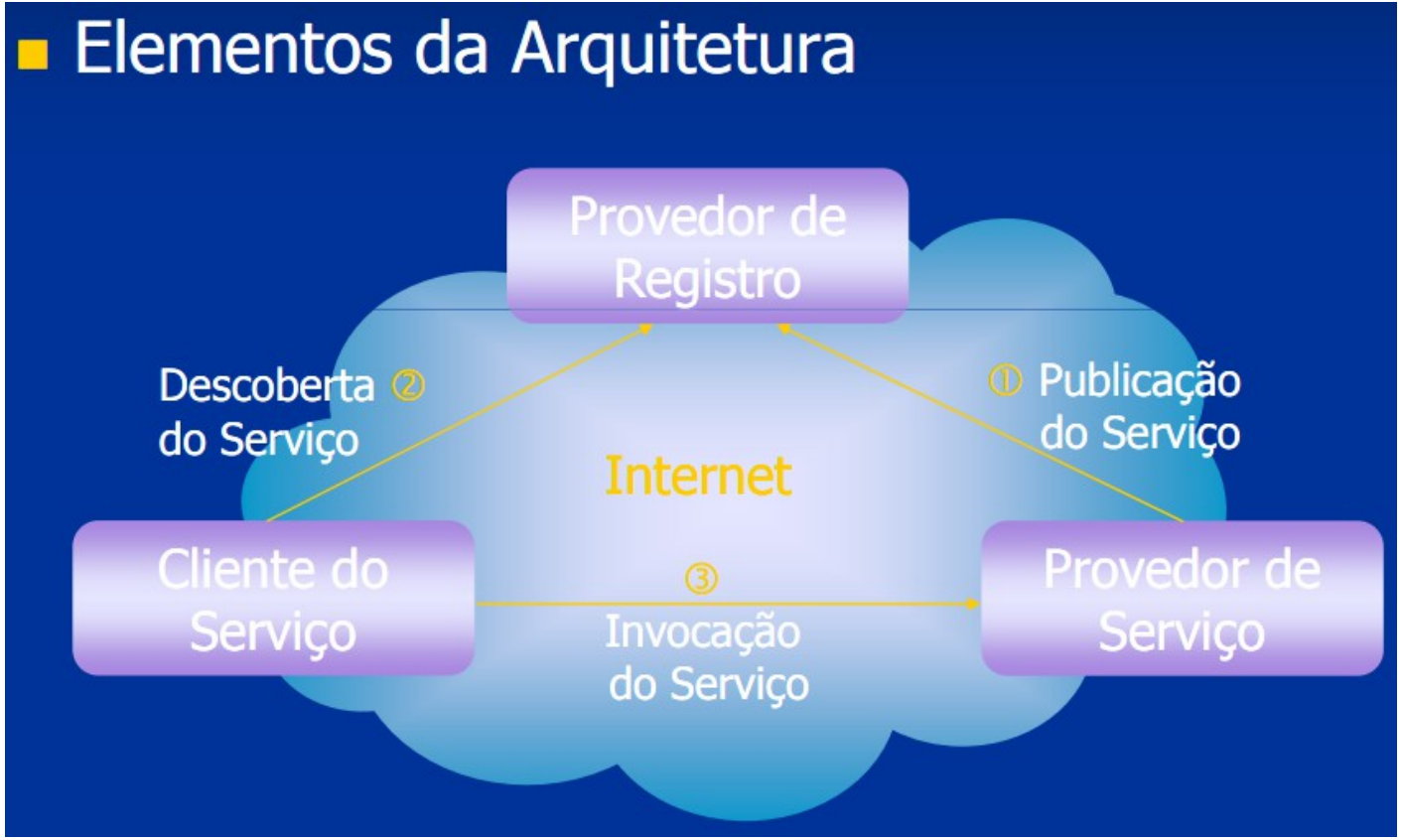
Método HTTP	Ação CRUD	Descrição	Códigos de Status
POST	CREATE	Cria um recurso	201, 400, 422
GET	RETRIEVE	Recupera uma representação do recurso	200, 301, 410
PUT	UPDATE	Cria ou atualiza um recurso	200, 301, 400, 410
DELETE	DELETE	Deleta um recurso	200, 204

- Respostas HTTP padronizadas:

HTTP Status Code	Meaning
1xx	Informational
2xx	Success
3xx	Redirection
4xx	Client Error
5xx	Server Error

200 - OK
201 - Created
204 - No Content
301 - Moved Permanently
400 - Bad Request
410 - Gone
422 - Unprocessable Entity

Como funciona os Web Services baseados no protocolo SOAP?


Elementos da Arquitetura
Provedor de Registro
Descoberta ② do Serviço
① Publicação do Serviço
Internet
Cliente do Serviço
③ Invocação do Serviço
Provedor de Serviço

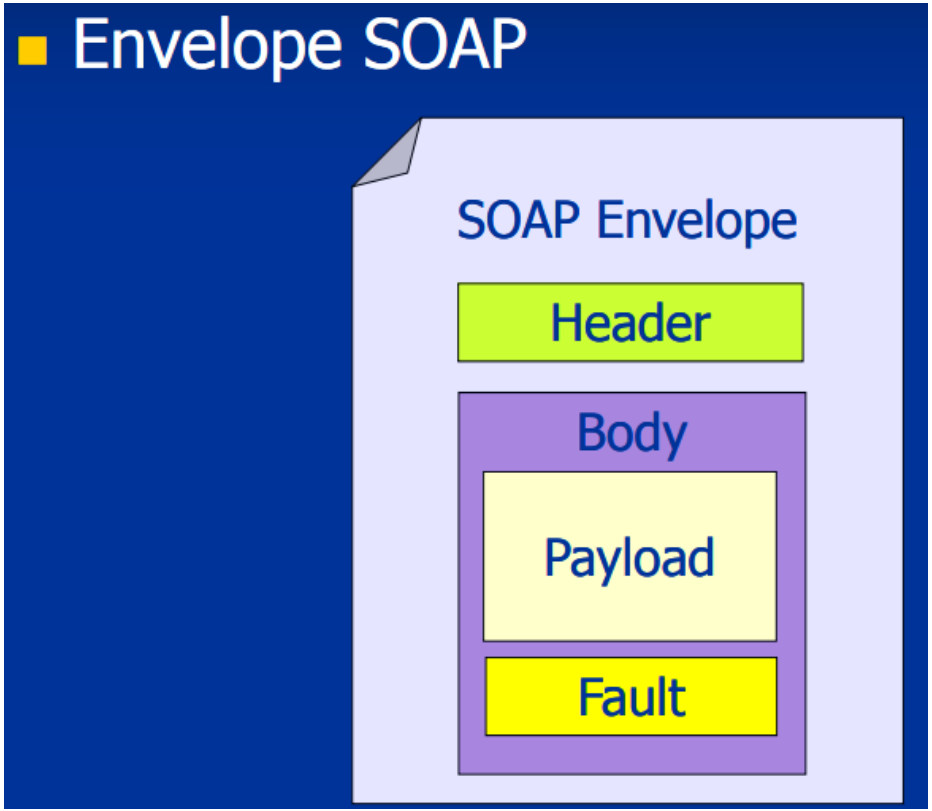
Web Services

- Tecnologias empregadas por Web Services
  - XML (*eXtensible Markup Language*): formato padrão para troca de dados
  - SOAP: protocolo utilizado na interação com os serviços Web
  - WSDL (*Web Services Description Language*): utilizada para descrever os serviços Web
  - UDDI (*Universal Description, Discovery and Integration*): permite localizar serviços na rede

- Protocolo SOAP
  - Protocolo definido pelo W3C para comunicação entre Web Services
  - Nome originou das iniciais de *Simple Object Access Protocol* (esse nome não é mais usado)
  - Define o formato das mensagens trocadas entre Web Services
  - Independente de plataforma e de linguagem
  - Utiliza em geral HTTP[S] como protocolo de transporte (porta 80 → atravessa *firewalls*)

- Funcionamento
  - Cliente cria um envelope SOAP especificando o nome da operação requisitada e os nomes e valores dos parâmetros da operação
  - Requisição é enviada pela rede ao provedor do serviço
  - Requisição é recebida e interpretada
  - A operação requisitada é executada
  - A resposta, se houver, é colocada em um envelope SOAP e enviada ao cliente

SOAP



Exemplo de Requisição SOAP

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
   <ns1:getTemperaturaMinima xmlns:ns1="http://ufsc.br/previsao">
      <localidade>Florianópolis</localidade>
   </ns1:getTemperaturaMinima>
</S:Body>
</S:Envelope>
```

Exemplo de Resposta SOAP

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
   <ns1:getTemperaturaMinimaResponse
    xmlns:ns1="http://ufsc.br/previsao">
      <return>13.2</return>
   </ns1:getTemperaturaMinimaResponse>
</S:Body>
</S:Envelope>
```

SOAP

- Linguagem de descrição de Web Services
  - Padrão do W3C
  - Baseado no XML
  - Especifica a interface de um serviço Web
  - Através do WSDL de um Web Service é possível saber que serviços estão disponíveis e como invocá-los remotamente
  - A especificação WSDL é independente da linguagem na qual o Web Service é implementado

Estrutura

```
<?xml version="1.0" encoding="UTF-8"?>
 <definitions name="PrevisaoDoTempo"
              targetNamespace="http://ufsc.br/previsao"
              xmlns:tns="http://ufsc.br/previsao"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:soap="http://www.w3.org/2003/11/wsdl/soap12"
              xmlns="http://www.w3.org/2003/11/wsdl">
+ <types>
+ <message name="getTemperaturaMinima">
+ <message name="getTemperaturaMinimaResponse">
+ <message name="getTemperaturaMaxima">
+ <message name="getTemperaturaMaximaResponse">
+ <portType name="Tempo">
+ <binding name="TempoPortBinding" type="tns:Tempo">
+ <service name="TempoService">
 </definitions>
```

- Elementos
  - <definitions>: elemento raiz
  - <types>: define os tipos de dados utilizados pelo serviço Web (pode referenciar um XSD)
  - <messages>: especifica as mensagens usadas na comunicação com o serviço Web
  - <portType>: define um conjunto de operações que são executadas por um serviço
  - <binding>: associa um protocolo ao serviço
  - <service>: especifica o endereço de rede no qual o serviço pode ser acessado

- Definição de Tipos
  - Importa um XSD com a descrição dos tipos

```
<types>
  <xsd:schema>
    <xsd:import namespace="http://ufsc.br/previsao"
     schemaLocation="http://ufsc.br/previsao/tempo.xsd" />
  </xsd:schema>
</types>
```

- Definição de Tipos em um XSD

```
<xs:schema xmlns:tns="http://ufsc.br/previsao"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           version="1.0" targetNamespace="http://ufsc.br/previsao">
<xs:element name="getTemperaturaMinima"
            type="tns:getTemperaturaMinima" />
<xs:element name="getTemperaturaMinimaResponse"
            type="tns:getTemperaturaMinimaResponse" />
<xs:complexType name="getTemperaturaMinima">
  <xs:sequence>
    <xs:element name="localidade" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="getTemperaturaMinimaResponse">
  <xs:sequence>
    <xs:element name="return" type="xs:float" />
  </xs:sequence>
</xs:complexType>
...
</xs:schema>
```

Definição de Mensagens

```
<message name="getTemperaturaMinima">
  <part name="parameters" element="tns:getTemperaturaMinima" />
</message>
<message name="getTemperaturaMinimaResponse">
  <part name="parameters"
        element="tns:getTemperaturaMinimaResponse"/>
</message>
<message name="getTemperaturaMaxima">
  <part name="parameters" element="tns:getTemperaturaMaxima" />
</message>
<message name="getTemperaturaMaximaResponse">
  <part name="parameters"
        element="tns:getTemperaturaMaximaResponse"/>
</message>
```

Definição de Porta

```
<portType name="Tempo">
  <operation name="getTemperaturaMinima">
    <input message="tns:getTemperaturaMinima" />
    <output message="tns:getTemperaturaMinimaResponse" />
  </operation>
  <operation name="getTemperaturaMaxima">
    <input message="tns:getTemperaturaMaxima" />
    <output message="tns:getTemperaturaMaximaResponse" />
  </operation>
</portType>
```

- *Binding* com o Protocolo SOAP

```
<binding name="TempoPortBinding" type="tns:Tempo">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
   style="document" />
  <operation name="getTemperaturaMinima">
    <soap:operation soapAction="" />
      <input> <soap:body /> </input>
      <output> <soap:body /> </output>
  </operation>
  <operation name="getTemperaturaMaxima">
    <soap:operation soapAction="" />
      <input> <soap:body /> </input>
      <output> <soap:body /> </output>
  </operation>
</binding>
```

Definição de Serviço

```
<service name="TempoService">
  <documentation>Serviço de Previsão do Tempo</documentation>
  <port name="TempoPort" binding="tns:TempoPortBinding">
    <soap:address location="http://ufsc.br/previsao/TempoService" />
  </port>
</service>
```

Agenda

1. URI, URL e HTTP
2. **O que é REST?**
3. Por que utilizar REST?
4. Restrições arquiteturais do REST
5. ‘Protocolo’ REST
6. REST x SOAP
7. Como organizar/gerenciar os recursos para web services REST
8. Arquitetura Orientada a Recursos (ROA – Resource Oriented Architecture)
9. Exemplos de serviços REST

REST

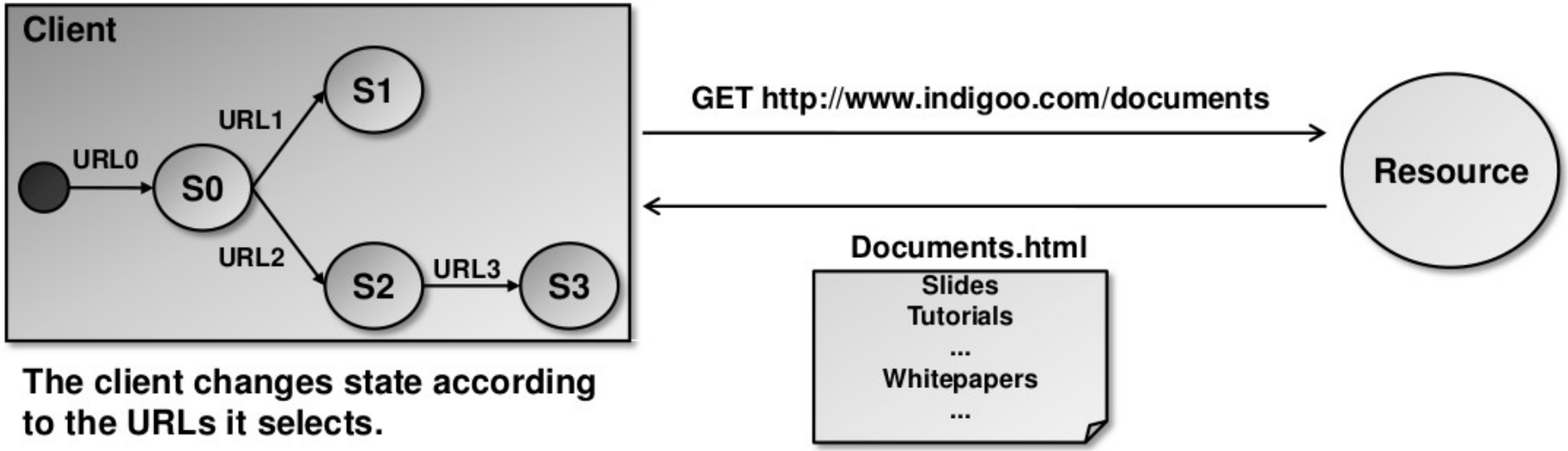
O que seria REST?

- REST, ao contrário do SOAP, não é um padrão de WS (web service), mas sim um estilo arquitetural para as aplicações Web
- REST foi concebido por Roy Fielding em sua tese de doutorado:
  - "Representation State Transfer is intended to evoke an image of how a well-designed Web application behaves: **a network of web pages** (a virtual state-machine), where the **user progresses through an application by selecting links** (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

- **REST não é um padrão ou protocolo**
  - REST é um estilo arquitetural.
- Faz uso de padrões Web existentes (HTTP, URL, XML, JSON, MIME types).
- É **orientado a recurso**
  - Os recursos (pedaços de informação) são endereçadas por URIs e passadas do servidor para o cliente (ou vice-versa)

Como acontece a navegação Web a partir de um navegador?

- Para entender o princípio REST, vamos analisar como funciona o acesso Web a partir de um navegador:



- Para entender o princípio REST, vamos analisar como funciona o acesso Web a partir de um navegador:
  1. O cliente acessa um recurso usando uma URL
  2. O servidor Web retorna uma **representação** do recurso na forma de um documento HTML
  3. Este recurso coloca o cliente em um novo **estado**
  4. O usuário clica em um link presente no recurso que resulta em outro acesso a recurso
  5. O novo recurso coloca o cliente em um novo estado

- A aplicação cliente troca **(= transfere) de estado** com cada representação do recurso

- REST baseia-se sobre os princípios e protocolos Web (WWW, HTTP):
  - **Recursos**:
    - O estado e a funcionalidade da aplicação são **abstraídas em recursos** (tudo é um recurso).
  - **Endereçamento dos recursos**:
    - Todo recurso é **unicamente endereçado** utilizando hyperlinks.
  - **Interface uniforme para acessar os recursos**:
    - Todos os recursos compartilham uma interface uniforme para a transferência de estado entre cliente e recurso, consistindo de:
      - um conjunto restrito (limitado) de operações bem definidas (GET, PUT, POST, DELETE).
      - um conjunto restrito de tipos de conteúdo(text/html, text/xml etc.).

Agenda

1. URI, URL e HTTP
2. O que é REST?
3. **Por que utilizar REST?**
4. Restrições arquiteturais do REST
5. ‘Protocolo’ REST
6. REST x SOAP
7. Como organizar/gerenciar os recursos para web services REST
8. Arquitetura Orientada a Recursos (ROA – Resource Oriented Architecture)
9. Exemplos de serviços REST

Por que utilizar REST?

Por que utilizar REST?

Por que utilizar REST?

- **Escalabilidade do WWW**:
  - O WWW tem provado ser:
    - escalável
    - simples (fácil de implementar, fácil de usar)
- **Motivação do REST**:
  - Se a Web é boa o suficiente para humanos, ele é boa o suficiente para interação machine-to-machine (M2M)
  - Os conceitos por trás do RPC-WS (SOAP, XML-RPC) são diferentes:
    - RPC-WS faz pouco uso dos conceitos e tecnologias WWW.
    - Tal WS define uma interface baseada em XML consistindo de operações que funcionam sobre o HTTP ou algum outro protocolo de transporte
    - Portanto, os recursos e capacidades do HTTP não são exploradas.

Por que utilizar REST?

- A motivação para o REST foi criar um **modelo de arquitetura** para web services que usam os mesmos princípios que fez o WWW ter sucesso.
- A meta é alcançar o mesmo **nível de escalabilidade e simplicidade**
  - REST utiliza **conceitos e tecnologias provadas**.
  - REST mantém as coisas o mais simples possível

Agenda

1. URI, URL e HTTP
2. O que é REST?
3. Por que utilizar REST?
4. **Restrições arquiteturais do REST**
5. ‘Protocolo’ REST
6. REST x SOAP
7. Como organizar/gerenciar os recursos para web services REST
8. Arquitetura Orientada a Recursos (ROA – Resource Oriented Architecture)
9. Exemplos de serviços REST

- REST define 6 restrições de arquitetura que a arquitetura do sistema deve cumprir para obter escalabilidade:
  - **Paradigma cliente-servidor**:
    - Um cliente recupera os recursos de um servidor ou atualiza dos recursos no servidor.
    - Separação dos conceitos tal como apresentação (cliente) do armazenamento de dados (servidor)
    - Portabilidade (UI pode ser portada para diferentes plataformas.
  - **Sem estado (Stateless)**:
    - Uma requisição cliente contém todas as informações necessárias para o servidor entender a requisição.
    - Sem necessidade de armazenar o contexto(estado) no servidor.
    - Melhor escalabilidade.

- REST - 6 restrições de arquitetura:
  - **Cacheable**:
    - Dados (recursos) necessitam ser rotulados como **cacheable ou não cacheable**.
    - Melhora o desempenho de rede
  - **Interface Uniforme:**
    - Interface uniforme significa que os **recursos são identificados de forma padrão** (uniforme). Além disso, os recursos são manipulados por **operações padrão**.
    - Uniformidade de interface entre componentes de uma aplicação distribuída é o recurso central do REST.
    - **Simplifica a arquitetura**
    - Desacopla a aplicação (serviço) da interface

- REST - 6 restrições de arquitetura:
  - **Sistema em camadas**:
    - Camadas visam a decomposição das funcionalidades do sistema
    - Encapsulamento das funcionalidades em camadas (Ex: encapsulamento de um serviço legado por trás de uma interface padrão)
    - Decomposição das funcionalidades do sistema em cliente, servidor e intermediário.
  - **Código sob demanda**:
    - Esta restrição é opcional para os sistemas estilo REST;
    - Código sob demanda significa o download e execução dinâmica do código no cliente (Javascript etc)
    - Extensibilidade (ex: extensão do cliente com código baixado do serviço).

Agenda

1. URI, URL e HTTP
2. O que é REST?
3. Por que utilizar REST?
4. Restrições arquiteturais do REST
5. **‘Protocolo’ REST**
6. REST x SOAP
7. Como organizar/gerenciar os recursos para web services REST
8. Arquitetura Orientada a Recursos (ROA – Resource Oriented Architecture)
9. Exemplos de serviços REST

‘Protocolo’ REST

- REST não é um protocolo como o SOAP.
  - Entretanto, REST define algumas características chaves que fazem o sistema REST-ful.
  - REST não é algo novo. Ele simplesmente utiliza padrões e protocolos existentes (HTTP, URI).
- Endereçamento de recursos:
  - REST utiliza URIs (atualmente URLs) para endereçar e nomear os recursos

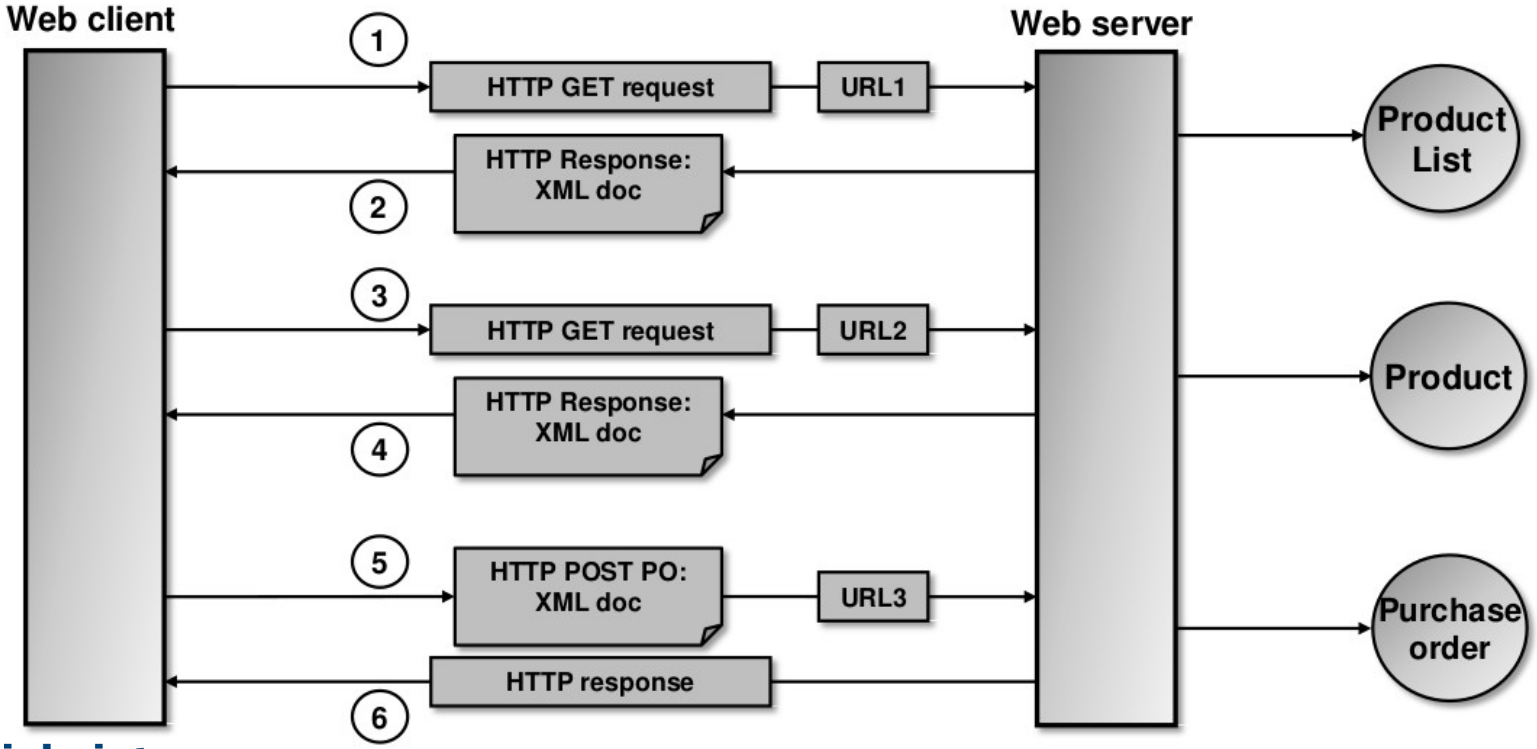
- Acesso aos recursos:
  - Ao contrário do RPC-WS onde o método de acesso (CRUD) é mapeado nas mensagens SOAP, REST utiliza os métodos HTTP como interface do recurso.

Create (C)	**→ HTTP POST**
Read (R)	**→ HTTP GET**
Update (U)	**→ HTTP PUT**
Delete (D)	**→ HTTP DELETE**

  - REST assume os métodos GET, HEAD, PUT, DELETE como idempotente
    - invocando o método múltiplas vezes sobre um recurso específico tem o mesmo efeito de invocá-lo uma única vez

- **Representação de recursos**:
  - REST utiliza representação de recurso padrão como HTML, XML, JSON, JPEG.
  - Representações comumente utilizadas são a XML e JSON
- **Tipos de dados**:
  - REST utiliza o campo **Content-type do HTTP header** (MIME types like text/html, text/plain, text/xml, text/javascript for JSON etc.) para indicar a codificação do recurso.
- **Estado**:
  - O **estado da aplicação** é para ser mantido no cliente.
  - O servidor **não mantém uma variável de estado para cada cliente** (isto melhora a escalabilidade).
  - O estado do recurso (criação, atualização ou remoção do recurso) é mantido no servidor

- Exemplo de acesso REST-ful:



- Exemplo de acesso REST-ful:
  - 1 e 2 - Requisição da lista de produtos:
    - O cliente requisita a lista de produtos que está disponível na URL http://www.cool-products.com/products&flavor=xml (URL1).
    - A resposta contém o recurso codificado em XML.
  - 3 e 4 - Seleção do produto:
    - A aplicação cliente (ou o usuário no navegador) seleciona o produto 00345 colocando em uma requisição para a URL http://www.cool-products.com/products/00345&flavor=xml (URL2).
    - A resposta contém uma representação XML para as informações do produto 00345.

- Exemplo de acesso REST-ful:
  - 5 e 6 - fazendo uma ordem de compra:
    - A aplicação cliente realiza uma ordem de compra para o produto 00345 requisitando o recurso na URL http://www.cool-products.com/products/00345/PO?quantity=7 (URL3).
    - A ordem de compra contém informações adicionais colocadas pelo cliente (nome do cliente, etc).
    - Dessa forma, a requisição é POST acompanhada de uma representação XML da ordem de compra.

Products list:

```
<?xml version="1.0"?>
<p:Products xmlns:p="http://www.cool-products.com"
            xmlns:xlink="http://www.w3.org/1999/xlink"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation=
                        "http://www.cool-products.com
                         http://www.cool-products.com/products.xsd">
    <Product id="00345" xlink:href="http://www.cool-products.com/products/00345"/>
    <Product id="00346" xlink:href="http://www.cool-products.com/products/00346"/>
    <Product id="00347" xlink:href="http://www.cool-products.com/products/00347"/>
    <Product id="00348" xlink:href="http://www.cool-products.com/products/00348"/>
</p:Products>
```

Product list contains links to get detailed information about each product (e.g. using XLink). This is a core feature of REST.

Product 00345 info:

```
<?xml version="1.0"?>
<p:Product xmlns:p="http://www.cool-products.com"
           xmlns:xlink="http://www.w3.org/1999/xlink"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation=
                      "http://www.cool-products.com
                       http://www.cool-products.com/product.xsd">
    <Product-ID>00345</Product-ID>
    <Name>Widget-A</Name>
    <Description>This product is made of disposable materials</Description>
    <Specification xlink:href="http://www.cool-products.com/products/00345/specification"/>
    <PurchaseOrder xlink:href=http://www.cool-products.com/products/00345/po/>
    <UnitCost currency="CHF">1.40</UnitCost>
    <Quantity>10</Quantity>
</p:Product>
```

Another URL is provided in the product XML response for placing purchase orders.

Agenda

1. URI, URL e HTTP
2. O que é REST?
3. Por que utilizar REST?
4. Restrições arquiteturais do REST
5. ‘Protocolo’ REST
6. **REST x SOAP**
7. Como organizar/gerenciar os recursos para web services REST
8. Arquitetura Orientada a Recursos (ROA – Resource Oriented Architecture)
9. Exemplos de serviços REST

SOAP x REST

Baseado no que foi visto, qual a diferença entre SOAP e REST?

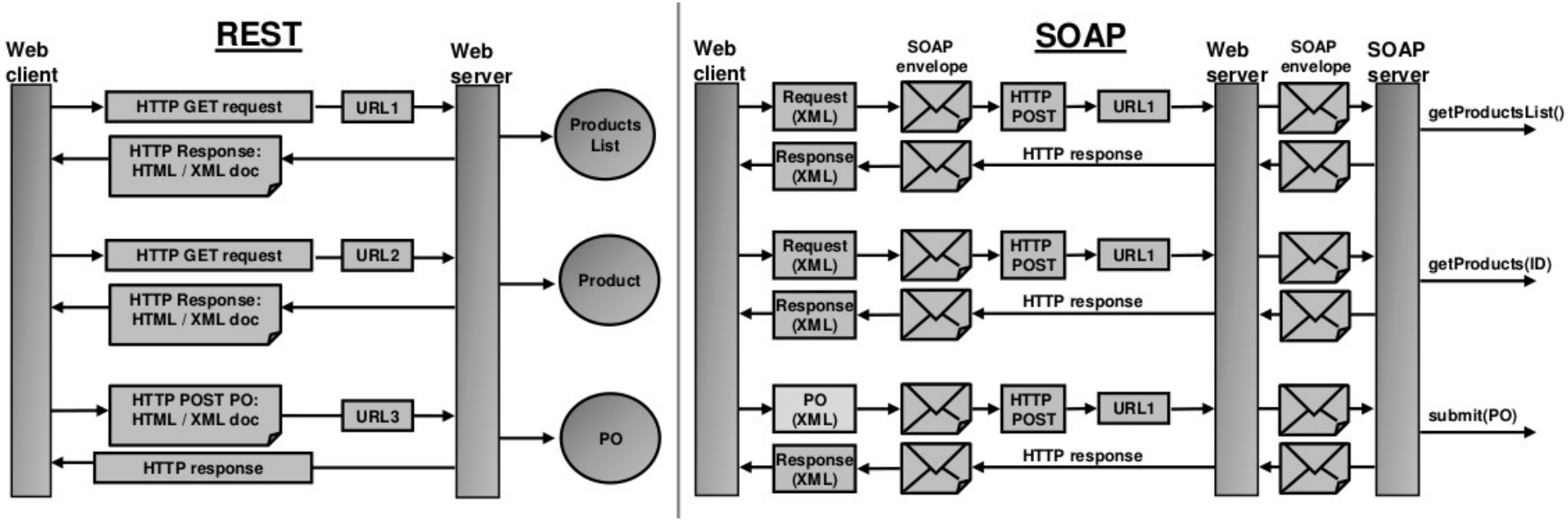
SOAP x REST

Aspecto	SOAP/WSDL(RPC-WS)	REST
Padronização	SOAP: W3C SOAP 1.2 WSDL: W3C WSDL 2.0	Sem padronização, faz uso dos padrões existentes como a RFC 2616 HTTP 1.1
Endereçamento do recurso	Indireto via operações SOAP	Todo recurso tem sua própria URL
URL	Somente utilizado para endereçar o processador SOAP	Usado para endereçar recursos individuais (conjunto de dados)
Apresentação dos dados	XML	Todas as codificações definidas pelo HTTP (XML, text, etc)
Uso do HTTP	Somente como um protocolo de transporte (envelope)	Ações sobre os recursos (CRUD) mapeados para métodos HTTP (PUT, GET, POST, DELETE)

SOAP x REST

Aspecto	SOAP/WSDL(RPC-WS)	REST
Estado	Stateful (toda ação SOAP é parte de uma interação definida pela aplicação)	Stateless (requisições são ‘autocontidas’, sem nenhum contexto armazenado no servidor)
Registro / descoberta de serviço	UDDI / WSDL	Nenhum (talvez mecanismos de busca como o Google consigam encontrar os registros de REST web services)
Métodos	Dentro do SOAP body	método HTTP
Escopo (quais dados?)	Dentro do SOAP body	Parte da URL

SOAP x REST

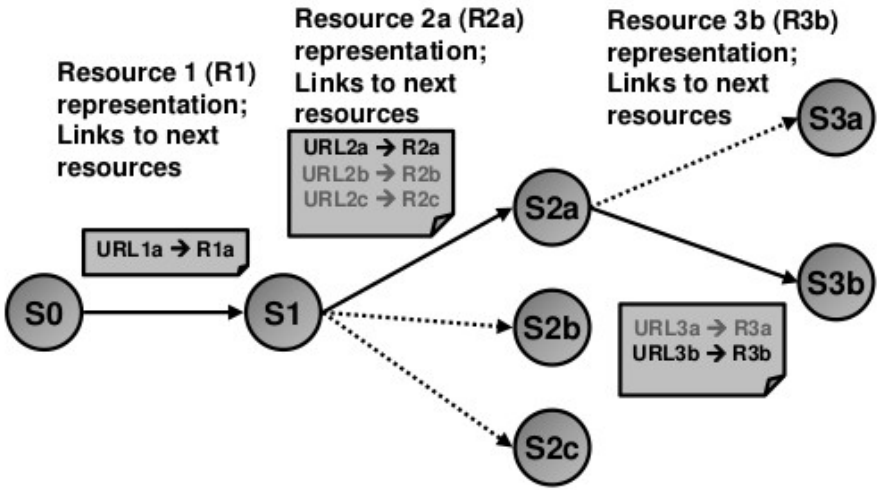


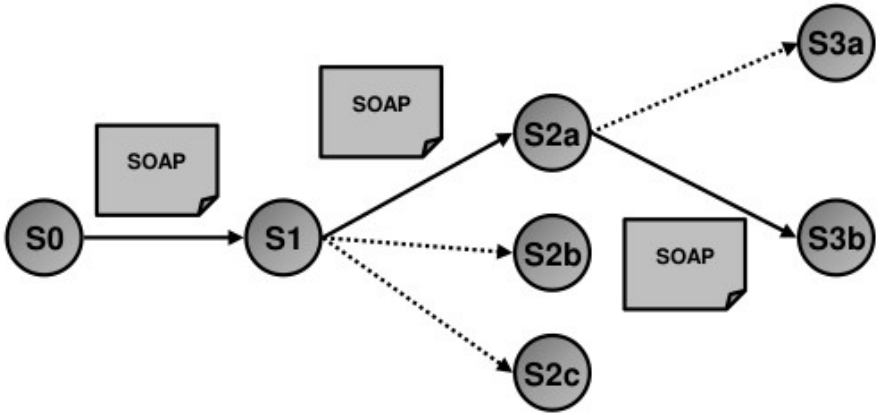
- REST utiliza diferentes URLs para endereçar os recursos
- O servidor Web diretamente dispara uma requisição para o handler (URL endereça o handler)
- REST mapeia o tipo de acesso aos métodos HTTP

- RPC-WS(SOAP) usa a mesma URL para todas as interações
- O servidor SOAP analisa a mensagem SOAP para determinar qual método chamar
- Todas as mensagens SOAP são enviadas como requisições HTTP POST

SOAP x REST

- REST é modelada depois do modelo de interação dos humanos com a web.
  - O usuário carrega uma página, ler e segue o link para uma outra página. Cada página o coloca em outro estado.
- REST aplica este modelo simples e coloca a maior parte do controle da aplicação no servidor (links nas respostas XML guiam através da aplicação.)

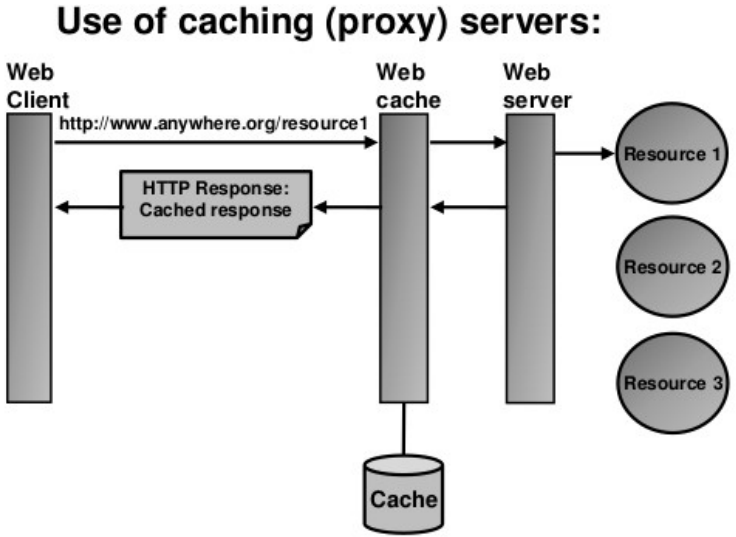


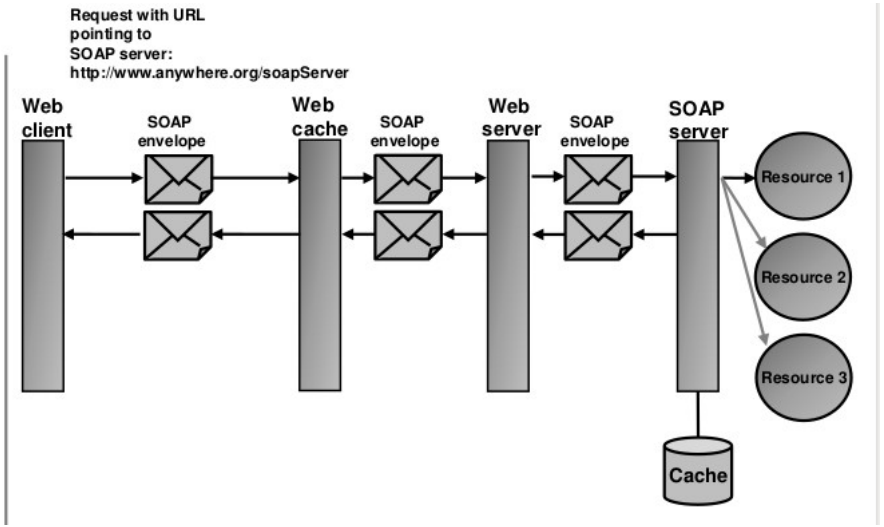


- Os links no documento XML de resposta ‘aponta’ para o próximo estado
- Utilizando a tecnologia XLink permite adicionar mais informação sobre o recurso relacionado (XLink:role)

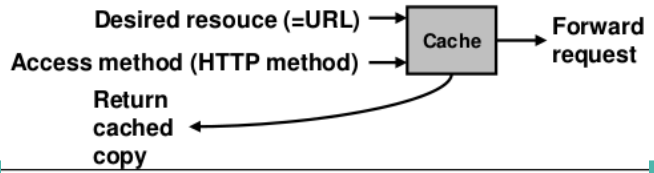
- As mensagens SOAP não contém hyperlinks, somente dados.
- O cliente não pode obter informação sobre o que fazer depois de receber a mensagem SOAP, mas deve pegar a informação em algum local.

SOAP x REST





- Caching em HTTP é uma tecnologia comprovada para melhorar os acessos e reduzir a carga de redes.
- REST pode (re)-usar a mesma lógica de cache



- O cache não determina diretamente se o recurso está em cache ou de pode ser recuperado de um cache
- Simples servidores cache não podem ser utilizados com SOAP
- Somente um servidor SOAP pode realizar cache, mas isto significa que ele já terá consumido os recursos de rede

Agenda

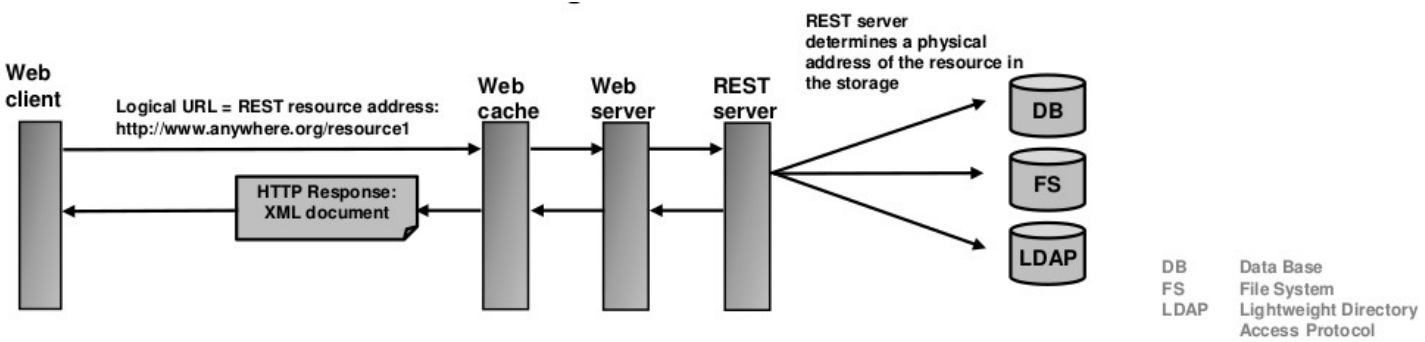
1. URI, URL e HTTP
2. O que é REST?
3. Por que utilizar REST?
4. Restrições arquiteturais do REST
5. ‘Protocolo’ REST
6. REST x SOAP
7. **Como organizar/gerenciar os recursos para web services REST**
8. Arquitetura Orientada a Recursos (ROA – Resource Oriented Architecture)
9. Exemplos de serviços REST

Como organizar recursos para web services REST

REST requer que todos os recursos necessários serem armazenados em arquivos XML individuais?

http://www.cool-products/products/000000
http://www.cool-products/products/000001
...
http://www.cool-products/products/999999

- Não é necessário.
  - REST utiliza URLs “lógicas”, isto é, endereços que identificam um recurso.
  - Um recurso é fisicamente armazenado em algum lugar (DB, file, directory service etc.).
- A implementação subjacente de um recurso e o seu armazenamento é transparente para um cliente REST
  - Recursos são convertidos para fragmentos XML “on-the-fly“, isto é, o servidor REST recupera os dados por exemplo de um DB e os converte em fragmentos XML



Agenda

1. URI, URL e HTTP
2. O que é REST?
3. Por que utilizar REST?
4. Restrições arquiteturais do REST
5. ‘Protocolo’ REST
6. REST x SOAP
7. Como organizar/gerenciar os recursos para web services REST
8. **Arquitetura Orientada a Recursos (ROA – Resource Oriented Architecture)**
9. Exemplos de serviços REST

Arquitetura Orientada a Recursos (ROA)

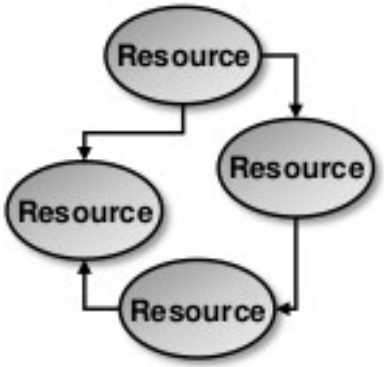
- ROA é similar ao SOA, mas utiliza o estilo REST web services ao invés do SOAP.
- **SOA -> SOAP WS**:
  - O serviço como uma coleção de tipos de operações e de mensagens é o ponto central do SOA.
  - Os dados (recursos) são acessíveis indiretamente via operações SOAP.
- **ROA -> REST WS**:
  - REST define um conjunto de critérios de projeto enquanto ROA é o conjunto de princípios de arquitetura.
  - Um recurso é uma unidade de dado que é relevante para ser endereçada, acessada e processada individualmente (Ex: um documento, uma linha no DB etc.).
  - Um recurso é identificado e endereçado por uma ou múltiplas URLs

Arquitetura Orientada a Recursos (ROA)

- Princípios do ROA:
  - **Endereçamento:**
    - Todos os dados são expostos como um recurso com uma URI.
  - **Statelessness**:
    - O contexto de acesso (ex: página de um resultado de uma busca) não deveria ser armazenada como um cookie (cookies são unRESTful).
    - Ao invés disso, o contexto / estado deveria ser modelado como uma URL
    - Exemplo:
      - /search?q=resource+oriented+architecture&start=50 (página é parte da URI).

- Princípios do ROA:
  - **Interface Uniforme**:
    - Mapeamento os métodos de requisição HTTP uniformemente (GET, PUT, DELETE, POST).
    - Um interface deveria ser segura e idempotente:
      - GET, HEAD -> segura (estado dos recursos e servidores não são alterados).
      - GET, HEAD, PUT, DELETE -> Idempotente (método pode ser chamado múltiplas vezes.
  - **Conectividade**:
    - Em ROA, os recursos devem estar conectados uns aos outros em suas representações

Arquitetura Orientada a Recursos (ROA)



Fully RESTful:
WS with addressable and connected resources.