FACULDADE DE TECNOLOGIA DE SÃO PAULO

WEB SERVICES (SOAP X REST)

SÃO PAULO
2012

FACULDADE DE TECNOLOGIA DE SÃO PAULO

WEB SERVICES (SOAP X REST)

Jean Carlos Rosário Lima
Monografia apresentada à Faculdade de Tecnologia de São Paulo
para a obtenção do Grau de Tecnólogo em Processamento de Dados
Orientador: Prof. Irineu Aguiar

SÃO PAULO
2012

RESUMO

Este trabalho tem como objetivo, demonstrar a tecnologia Web Services, sua importância e popularidade. Demonstrando assim as tecnologias que são utilizadas dentro do cenário dos Web Services, assim como as novas tendências que surgiram com o amadurecimento dos sistemas distribuídos. Dentre as tecnologias que serão abordadas neste trabalho, temos: XML, SOAP ,WSDL e UDDI. Por fim, analisaremos a abordagem REST em oposição ao protocolo SOAP de forma a demonstrar vantagens e desvantagens na utilização de ambas as abordagens.

ABSTRACT

This paper aims to demonstrate the Web Services technology, its importance and popularity. Thus demonstrating the technologies that are used within the scenario of Web services, as well as new trends that have emerged with the maturation of distributed systems. Among the technologies that will be addressed in this work, we have: XML, SOAP, WSDL and UDDI. Finally, we analyze the REST approach as opposed to the SOAP protocol in order to demonstrate advantages and disadvantages in the use of both approaches.

LISTA DE ABREVIATURAS E SIGLAS

REST	Representational State Transfer
ROA	Arquitetura Orientada a Serviço
CORBA	Common Object Request Broker Architecture
WSDL	Web Services Description Language
W3C	World Web Wide Consortium
UDDI	Universal Description Discovery and Integration
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
XML	Extensible Markup Language
URI	Uniform Resource Identifier
XSD	XML Schema Definition

SUMÁRIO

INTRODUÇÃO

A utilização da tecnologia de Web Services vem se popularizando nos últimos anos. Com o aumento da confiança do consumidor para realizar transações pela **INTERNET**, as empresas estão utilizando cada vez mais essa tecnologia.

Os Web Services são bastante utilizados em aplicações de comércio eletrônico. Um bom exemplo é a integração do comércio eletrônico com o serviço dos correios, o qual possibilita que o usuário informe o número do CEP e o serviço dos correios se encarrega de retornar o respectivo endereço para o CEP informado.

O XML é a tecnologia responsável por essa interoperabilidade, conectando programas e aplicações desenvolvidas em diferentes linguagens ou plataformas. (EXTENSIBLE, 2011)

O objetivo deste trabalho é demonstrar a integração da tecnologia de Web Services em sistemas heterogêneos nas suas diferentes abordagens: SOAP E REST.

I Web Services

I.1 Origem

Com a evolução das redes de computadores surgiram as aplicações distribuídas. Inicialmente todo o processamento era centralizado em apenas um servidor. Com o surgimento dos middlewares o processamento começou a ser distribuído entre vários servidores. Com o avanço da internet e dos protocolos de comunicação baseados em XML, surgiram os Web Services com a missão de integrar sistemas heterogêneos. (GOMES, 2010)

I.2 Definição

Segundo definição do W3C, os Web Services são aplicações autocontidas, que possuem interface baseadas em XML e que descrevem uma coleção de operações acessíveis através da rede, independente da tecnologia usada na implementação do serviço. (WEB, 2011)

O XML é o formato de mensagem adotado pelo W3C para troca de informações entre aplicações distribuídas através do protocolo HTTP. O SOAP por ser uma tecnologia sem características proprietárias tornou-se uma alternativa aos protocolos tradicionais, tais como: CORBA e DCOM. (GOMES, 2010)

Os Webs Services devem ser definidos de forma consistente para que possam ser descobertos e interfaceados com outros serviços e aplicações. A WSDL é uma especificação W3C que fornece a linguagem mais avançada para a descrição de definições de Web Services. A figura 1 abaixo mostra a definição de um documento WSDL.

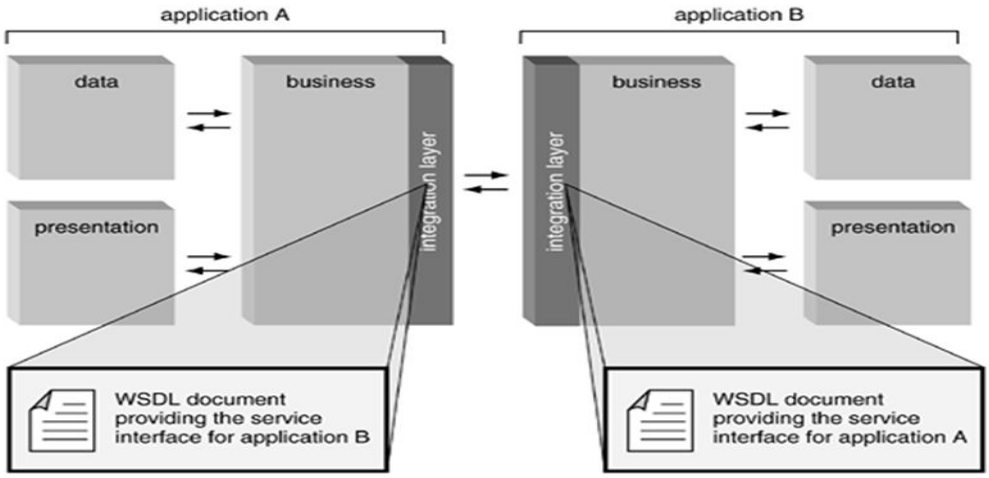

Figura 1-Documentos WSDL representando aplicações Web Services

A camada de integração introduzida pela estrutura de Web Services estabelece um padrão, universalmente reconhecido e com interface suportada. Tal como mostrado na **figura 1**, a WSDL permite a comunicação entre essas camadas ao fornecer descrições padronizadas.

Simplificadamente, pode se dizer que o arquivo WSDL é um documento XML que descreve um conjunto de mensagens SOAP e a forma como essas mensagens são trocadas.

Para enxergar o valor do WSDL, imagine que você quer invocar um método SOAP que é fornecido por um dos seus parceiros de negócio. Você pode pedir alguns exemplos de mensagens SOAP e escrever sua aplicação para produzir e consumir mensagens que se parecem com os exemplos fornecidos, mas isso pode gerar muitos erros. Por exemplo, você pode assumir que um campo é um inteiro, quando de fato o mesmo é uma string. O WSDL especifica o que a mensagem de requisição deve conter e como vai ser a resposta, em uma notação não ambígua.

A notação que o arquivo WSDL usa para descrever o formato das mensagens é baseada no padrão XML, o que significa que é uma linguagem de programação neutra, baseada em padrões. O que a torna adequada para descrever as interfaces dos Web Services, que são acessíveis por uma grande variedade de plataformas e linguagens de programação. Além de descrever o conteúdo das mensagens, o WSDL define onde o serviço está disponível e quais protocolos de comunicação são usados para conversar com o serviço. Isso significa que o arquivo WSDL define tudo que é necessário para escrever um programa que utilize o XML Web Service.

UDDI é o responsável pela publicação e descoberta dinâmica dos serviços Web Services. Para fazer uma chamada a um Web Service é necessário localizá-lo e em seguida descobrir sua interface e a semântica da sua chamada e escrever e configurar o software local que realizará a chamada ao Web Service. (UNIVERSAL, 2011)

UDDI são as páginas amarelas dos Web Services. Assim como nas páginas amarelas tradicionais, você pode procurar por uma companhia que ofereça os serviços que você precisa ler sobre o serviço oferecido e contatar alguém para obter mais informações sobre o serviço disponibilizado. Você pode naturalmente oferecer um serviço sem registrá-lo na UDDI, assim como você pode abrir um negócio no porão de sua casa e contar com a propaganda boca a boca, mas se você quiser alcançar um mercado significativo, você precisará do UDDI, assim seus clientes poderão encontrá-lo.

Um diretório UDDI é um arquivo XML que descreve o negócio e os serviços. O mesmo possui três partes:

- Páginas brancas: Descrevem a empresa (Nome, Endereço, Contatos).
- Páginas amarelas: Incluem as categorias, baseadas em taxonomias padrões.
- Páginas verdes: Descreve a interface para o serviço, em nível de detalhe suficiente para escrever uma aplicação que use Web Service.

O diretório UDDI também inclui várias maneiras de procurar os serviços. Por exemplo, pode se procurar fornecedores de um determinado serviço em uma região específica.

I.3 Papéis

Há três papéis importantes dentro da arquitetura de Web Services: Provedor de Serviços, Consumidor de serviços e o Registro dos serviços. A interação destes papéis envolve as operações de publicar, pesquisar e fazer a ligação dos serviços. Abaixo é definida a função para cada um destes papéis:

- Provedor de serviços: É o responsável pela implementação e disponibilização dos Web Services na INTERNET. Para que alguém possa utilizá-lo, é preciso descrever o serviço em um formato padrão, compreensível para qualquer um que precise usar esse serviço e também publicar as características sobre o serviço em um registro central disponível.

- Consumidor de serviços: É o usuário de um serviço disponível na INTERNET que foi implementado por um provedor de serviço.

- Registro do serviço: Refere-se à localização do serviço. Ele contém as informações técnica dos serviços e os detalhes da empresa.

A figura 2 ilustra a interação entre os elementos da arquitetura Web Services.

Figura 2 - Interação entre os elementos da arquitetura de Web Service

O provedor é o responsável pela publicação da descrição de um determinado serviço. O Consumidor por sua vez, faz a utilização de tal descrição para encontrar o serviço disponibilizado pelo provedor.

As comunicações entre as aplicações de Web Services fazem uso de quatro elementos que fazem o encapsulamento da requisição e resposta entre um servidor e um cliente. Estes elementos são:

- XML
- SOAP
- WSDL
- UDDI

A figura abaixo ilustra como esses elementos se relacionam.

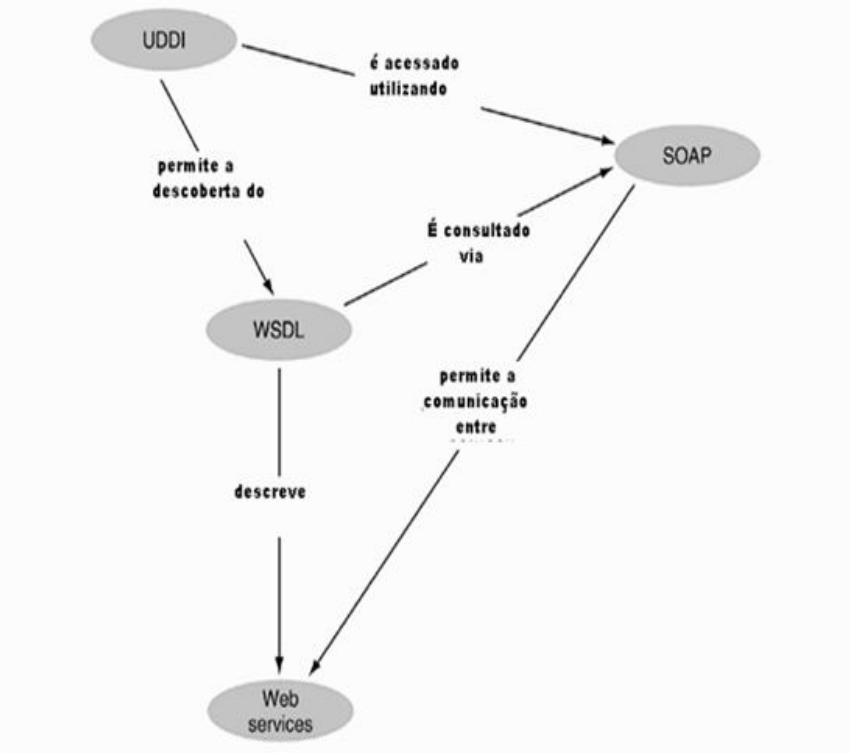

Figura 3 - Relacionamento entre as especificações

II RPC

II.1 O que é RPC ?

O RPC é uma técnica utilizada no desenvolvimento de aplicações distribuídas que segue o modelo cliente-servidor. (REMOTE, 2011)

O mesmo tem por finalidade permitir que um procedimento local possa ser estendido, ou seja, que o procedimento chamado possa existir em um espaço de endereçamento remoto.

O RPC trabalha com o modelo cliente-servidor. Ambos podem estar no mesmo computador ou em computadores diferentes conectados através uma rede.

Com o uso de RPC, um programador de uma aplicação distribuída pode evitar ter que tratar de detalhes de baixo nível impostos pela interface com a rede.

Primeiras implementações comerciais:

- ONC RPC (também conhecida como SUNRPC
- Disponível na maioria dos sistemas UNIX
- Código aberto disponível desde 1988

II.2 Como funciona ?

Um procedimento remoto é identificado de forma unívoca pela trinca:

- Número do Programa: Identifica um grupo de procedimentos remotos relacionados. Sendo que cada programa possui um número de procedimento diferente.
- Número da Versão: Um programa pode consistir de uma ou mais versões.

- Número do Procedimento: Identifica o programa em uso.

Quando o servidor inicia, ele registra-se através do serviço **portmapper.** Ele também registra os números de programa e de versão dos procedimentos que disponibiliza. Antes que o cliente possa fazer uma chamada RPC, ele consulta o **portmapper** do servidor para identificar a porta em que o servidor está ativo para realizar a comunicação. Por fim, o cliente e servidor estabelecem um canal de comunicação.

Após o canal ter sido estabelecido. A chamada RPC funciona da seguinte forma:

1. O cliente faz uma chamada que envia uma requisição ao servidor e aguarda a resposta.

2. O cliente é bloqueado até o recebimento de uma resposta ou se o tempo de espera (timeout) se esgotar.

3. Quando a requisição é enviada ao servidor, o mesmo chama um procedimento que fornece o serviço solicitado e envia a resposta de volta para o cliente.

4. Após o recebimento da resposta pelo cliente, este continua a sua execução.

II.3 Interface RPC

A interface de programação RPC é dividida em níveis. Esses níveis oferecem diferentes possibilidades de controle sobre os detalhes envolvidos na comunicação entre o cliente e o servidor. Em outras palavras, quanto maior for o controle desejado, maior será a quantidade de código necessária. Existem cinco níveis:

- Simples: Realiza chamadas de procedimento e registra o servidor

- Alto: Além de realizar chamadas. É possível configurar cliente e servidor.

- Intermediário: Especifica o tipo de transporte.

- Especialista: Especifica detalhes do transporte.
- Baixo: Tem total controle sobre o transporte.

O nível simples é composto basicamente por duas funções: **callrpc** e **registerrpc.** A função **callrpc** é responsável por fazer a chamada do procedimento remoto no lado do cliente e recebe os seguintes parâmetros:

- Nome do Servidor
- Número do Programa
- Número do Procedimento
- Filtro XDR: Responsável por codificar o argumento; ponteiro para o argumento. E decodificar o retorno e endereço para o retorno.

Já o **registerrpc** é responsável por realizar o registro do procedimento no lado do servidor, o mesmo recebe os seguintes parâmetros:

- Número do Programa
- Número da Versão
- Número do Procedimento
- Nome do Procedimento
- Filtro XDR: Responsável por codificar o argumento e decodificar o retorno.

Essas funções escondem todos os detalhes de rede, impedindo o controle sobre diversos detalhes da comunicação. Como por exemplo, o tipo de transporte utilizado.

II.4 Desenvolvendo RPC

Na maioria dos casos, não é necessário conhecer além do nível simples da interface RPC, para que se construa um serviço do tipo RPC. Através de programas auxiliares é possível gerar a interface de forma automática, sem a necessidade de codificar. O programador precisa apenas implementar o código da aplicação.

Para desenvolver uma aplicação RPC os seguintes passos são necessários:

1. Especificar o protocolo para a comunicação cliente/servidor:

- A maneira mais fácil para definir o protocolo é criar um arquivo usando uma linguagem especial.
- Nesse arquivo deve-se definir o nome dos procedimentos, os parâmetros dos procedimentos e os tipos de retorno.
- O compilador do protocolo lê esse arquivo e automaticamente gera grande parte do código necessário para a programação da aplicação RPC

2. Programar o cliente e o servidor.

3. Compilar os programas.

4. Disparar os serviços necessários.

   - Para que uma aplicação RPC funcione é necessário que o serviço **portmapper** esteja em execução.

5. Executar os programas.

III SOAP

III.1 Conceito

O SOAP é um protocolo projetado para invocar aplicações remotas através de RPC ou trocas de mensagens, em um ambiente independente de plataforma e linguagem de programação. (SIMPLE, 2011)

É fácil de implementar, testar e usar o protocolo SOAP. Ele é um padrão da indústria e foi adotado pelo W3C. A mensagem é transportada através do protocolo HTTP por meio de pacotes virtualmente idênticos .Os protocolos de autenticação e encriptação são os mesmos. O SOAP beneficia-se da infraestrutura criada para o HTTP para atravessar "firewalls" e roteadores, o que facilita a comunicação entre os Web Services envolvidos.

Inicialmente o SOAP foi construído como um veículo genérico para troca de mensagens entre computadores através da INTERNET. O formato de mensagem refere-se ao estilo utilizado entre as aplicações Web Services. O estilo de comunicação a ser utilizado entre um cliente (Web Services) e um serviço vai depender da forma que o serviço foi implementado.

Os Web Services oferecem dois tipos de modelos:

- RPC
- Document.

O RPC permite modelar chamadas de métodos com parâmetros e receber valores de retorno. Uma mensagem SOAP leva em seu corpo (elemento Body), o nome do método a ser executado e os parâmetros de entrada da chamada. Na mensagem RPC de resposta, um valor de retorno ou uma falha do método invocado é retornado.

No Document, o corpo da mensagem SOAP contém um fragmento de documento XML que é enviado ao serviço em vez de um conjunto de valores de parâmetros.

O modelo utilizado para a comunicação SOAP nos Web Services, influi diretamente em seu desempenho e em sua confiabilidade. Web Services implementados no estilo *RPC* necessita somente da elaboração da interface de suas operações, enquanto que no estilo *Document* exige mais esforço de implementação, pois requer a criação de um XML Schema, o qual necessita da descrição dos elementos que correspondem às operações do serviço e os respectivos tipos de dados utilizados nos parâmetros das operações.

III.2 Características

- Definido pelo consorcio W3C.
- Protocolo baseado em XML para troca de informações em ambientes distribuídos
- Padrão de utilização para Web Services.
- Normalmente utiliza HTTP como protocolo de transporte.

III.3 Formato de Mensagem

Uma mensagem SOAP consiste basicamente dos seguintes elementos:

Figura 4 - Elementos da mensagem SOAP

- Envelope: É o elemento principal do XML que representa a mensagem.
- Header: É um mecanismo genérico que permite a adição de características à mensagem SOAP de forma descentralizada, sem acordo anterior entre as partes comunicantes.
- Body: Este elemento contém a informação a ser transportada.

III.4 Especificação do Protocolo

A especificação do protocolo SOAP está dividida em quatro partes:

- SOAP Envelope: Define o que há na mensagem e como processá-la. É a única parte do protocolo que é obrigatória.
- SOAP Encoding Rules: Define um mecanismo de serialização que pode ser usado para troca de instâncias de tipos definidos pela aplicação.
- SOAP RPC Style: Define uma convenção que pode ser usada para representar as chamadas e respostas remotas aos procedimentos.
- LINK SOAP HTTP: Define um protocolo que liga o SOAP e o HTTP. Ele descreve como as mensagens SOAP são transmitidas via HTTP.

III.5 Envelope SOAP

Este elemento é parte obrigatória de uma mensagem SOAP. Ele funciona como um recipiente que comporta os demais elementos da mensagem, tais como: O cabeçalho (Header), o corpo (Body) e os respectivos *namespaces* de cada um. Da mesma forma que o correio entrega uma carta baseando-se pelo nome e endereço do destinatário, o elemento

envelope precisa destas informações do protocolo de transporte que está ligado a ele, com o propósito de garantir o envio da mensagem para o local correto. O Envelope pode possuir um cabeçalho que se chama SOAP ACTION que indica o endereço de entrega da mensagem. Um dos principais motivos para inserir o cabeçalho é para que os administradores de sistemas possam configurar seus firewalls para filtrar as mensagens baseadas nas informações dos cabeçalhos.

A figura 5 tem um exemplo de formatação do Envelope SOAP

```
<?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">
  ...
  Message information goes here
  ...
</soap:Envelope>
```

Figura 5 - Envelope SOAP

III.6 Cabeçalho SOAP

Este elemento é opcional em uma mensagem SOAP. Ele contém informações adicionais, como por exemplo, se a mensagem deve ser processada por um determinado nó intermediário. É importante lembrar que ao trafegar pela rede, a mensagem passa por diversos pontos intermediários até alcançar o seu destino final. Quando utilizado, o cabeçalho deve ser o primeiro elemento do Envelope.

```
<?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:Header>
  <m:Trans xmlns:m="http://www.w3schools.com/transaction/"
  soap:mustUnderstand="1">234
  </m:Trans>
</soap:Header>
...
...
</soap:Envelope>
```

Figura 6 - Cabeçalho SOAP

III.7 Atributos SOAP

Existem três atributos no SOAP que podem ser utilizados na composição de uma mensagem:

- **Atributo *encodingStyle***: indica as regras deserialização usadas nas mensagens SOAP. Podendo aparecer em qualquer elemento, o seu escopo é todo o conteúdo do elemento, assim como os elementos filhos. O valor deste atributo é uma lista de uma ou mais URIs , os quais identificam as regras de serialização ou deserialização, indicadas na ordem da mais específica para a mais abrangente. Exemplos de valores para o atributo *encodingStyle.*

Figura 7 - Valores válidos para o atributo Style

A terceira linha, corresponde a um valor vazio, indicando que não há requisição de *encodingStyle* para o elemento contido. Pode ser utilizado para "decodificar" uma requisição feita a um elemento pai.

- **Atributo *actor:*** algumas aplicações SOAP são chamadas intermediárias, pois tem a possibilidade de encaminhar e receber a mensagem SOAP. Uma mensagem SOAP pode sair de um remetente para um destinatário e, potencialmente, passar por várias aplicações SOAP intermediárias. Nem todas as partes de uma mensagem SOAP interessam a um destinatário, assim como podem interessar a um ou mais intermediários no caminho da mensagem. Quando uma aplicação SOAP "A" recebe um elemento de cabeçalho, ela é dita *receptora* e encara o cabeçalho como um contrato entre ela e quem repassou a mensagem. Assim ao enviar a mensagem para

outra aplicação “B”, a aplicação “A” deve incluir um cabeçalho similar ao atual, mas no caso, o contrato deve ser entre “A” e “B”. O atributo *actor* pode ser utilizado como um indicador de receptor de um elemento de cabeçalho. Funciona como um meio de *hops* representado pelo campo *Connection* do cabeçalho do HTTP. O valor é uma URI, que se for vazio, designa o receptor como o destinatário.

- **Atributo *mustUnderstand* :** Este atributo pode ser utilizado para indicar se uma entrada do cabeçalho é obrigatória ou opcional para o receptor processar.

III.7 Corpo SOAP

Este elemento é obrigatório na mensagem SOAP, o corpo (Body) armazena os dados de uma chamada de um método em particular, como o nome do método e parâmetros de entrada e saída e o respectivo resultado produzido pelo método. Ele está presente logo após o cabeçalho (Header) da mensagem, se este existir. Caso não informado, deverá aparecer imediatamente após a t*ag* de abertura do envelope.

O conteúdo do corpo da mensagem SOAP depende se ela é uma requisição ou uma resposta. Se for uma requisição, ele contém informações sobre a chamada do método, se for uma resposta irá conter os dados do resultado da chamada ao método.

Nas figuras abaixo. Temos uma demonstração de uma requisição, utilizando o protocolo SOAP. Na figura 8 é informado o valor do parâmetro do método GETPRICE. Já na figura 9 temos o retorno da requisição, ou seja, é retornado o preço do item informado.

```
<?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>
  <m:GetPrice xmlns:m="http://www.w3schools.com/prices">
    <m:Item>Apples</m:Item>
  </m:GetPrice>
</soap:Body>

</soap:Envelope>
```

Figura 8 - Exemplo de uma requisição SOAP

```
<?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>
  <m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
    <m:Price>1.90</m:Price>
  </m:GetPriceResponse>
</soap:Body>

</soap:Envelope>
```

Figura 9 - Resultado da requisição

A especificação SOAP define os tipos de dados baseados no *XSD*. Esta especificação define os tipos primitivos de dados, assim como estruturas mais complexas. Uma grande vantagem do uso do protocolo SOAP é que ele aceita qualquer tipo de dado que possa ser representado por um XSD Schema. Na tabela 1 temos os tipos aceitos pelo XSD.

Boolean	float	TimeInstan
Byte	Int	UnsignedByte
Doublé	Long	unsignedInt
Datatype	qname	unsignedLong
Decimal	short	unsignedShort
Enumeration	string	

Tabela 1 - Tipos de dados básicos suportados pelo SOAP

III.8 Tratamento de Exceções SOAP

Os métodos acessados através do protocolo SOAP não estão livres de erros, por isso necessitam de um dispositivo para identificá-los. Estes erros ou exceções que ocorrem nos *Web Services* devem ser retornados ao consumidor de alguma maneira, assim o Método de Exceção (Fault) executa esse trabalho. As exceções SOAP podem acontecer em vários estágios durante o processamento de uma requisição a um serviço.

Se a falha acontecer durante o transporte da mensagem, a camada do HTTP será responsável pela notificação do erro. Se o erro ocorrer na própria execução do protocolo SOAP, o mesmo tratará a exceção gerada.

IV REST

Foi idealizado por Roy Fielding no ano de 2000, na sua dissertação de doutorado, na qual buscou as melhores práticas nos estilos de arquiteturas existentes para compor um novo estilo que as reunissem em apenas um estilo, o qual ficou conhecido como REST.

REST é um estilo de arquitetura direcionado para sistemas de hipermídia distribuídos. Basicamente esse estilo é composto por dois papéis: Cliente e Servidor. Conforme figura abaixo.

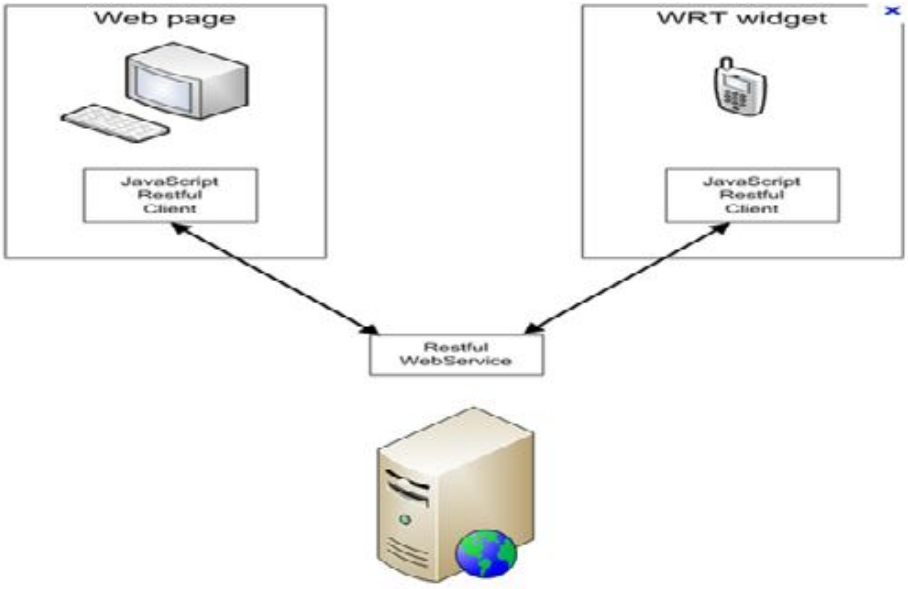

Figura 10 - Consumindo um serviço REST

O REST é basicamente o resultado das melhores práticas dos seguintes estilos:

- Cliente / Servidor.
- Sistema em Camadas
- Cache
- Sem estado

IV.1 Cliente/ Servidor

Neste modelo são definidos dois papéis: Cliente e Servidor. Basicamente o servidor disponibiliza um conjunto de serviços e o cliente faz uso desses serviços.

O cliente envia requisições para o servidor, este por sua vez ao recebê-las toma a decisão de aceitá-las ou não.

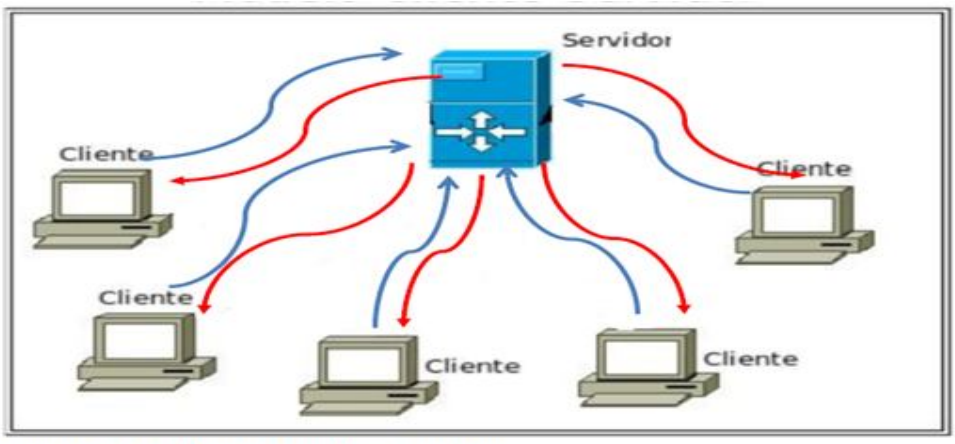

Figura 11 - Modelo Cliente - Servidor

Clientes e Servidores comunicam-se através de uma rede de computador com hardwares separados, porém o cliente e o servidor podem residir no mesmo sistema. A máquina servidora é um host que está executando um ou mais programas que compartilham os seus recursos com os clientes. O cliente por sua vez não compartilha recursos, ele solicita serviços ao servidor.

Funções como a troca de e-mail, acesso à INTERNET e acesso ao banco de dados, são construídos com base no estilo cliente-servidor. Por exemplo, um navegador da web é um programa cliente em execução no computador do usuário que pode acessar informações armazenadas em um servidor web na INTERNET.

IV.2 Camadas

No sistema em camadas, o mesmo é divido em camadas, onde cada camada conhece apenas a interface da camada superior. As camadas intermediárias podem ser utilizadas para melhorar a escalabilidade do sistema, permitindo o balanceamento de carga de serviços, através de múltiplas redes. (FIELDING,2000)

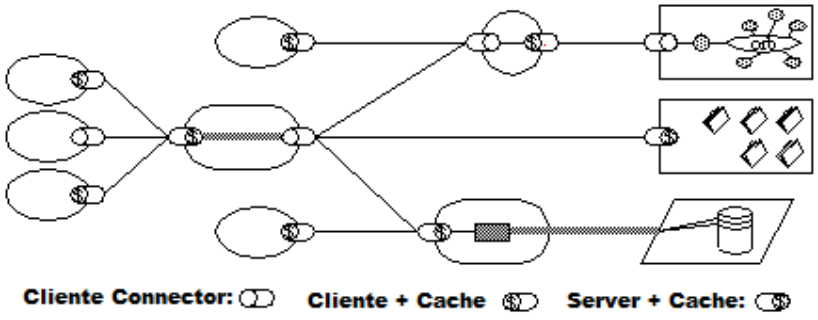

Figura 12 - Sistema em camadas

A desvantagem do sistema em camadas é que eles adicionam sobrecarga ao tratamento de dados. Essa perda de desempenho pode ser compensada se a rede for configurada para suportar cache.

IV.3 Cache

A arquitetura cache evita desperdício de banda, pois evita que dados que já tenham sido enviados anteriormente ao cliente sejam reenviados. Na primeira vez que uma página é solicitada pelo cliente a mesma é armazenada num Proxy HTTP.((FIELDING,2000)

A vantagem de se utilizar cache é a eliminação parcial ou total de algumas interações entre cliente e servidor, o que melhora a eficiência (menos trafego de rede) , escalabilidade (menos processamento) e performance , já que o servidor fica menos carregado.

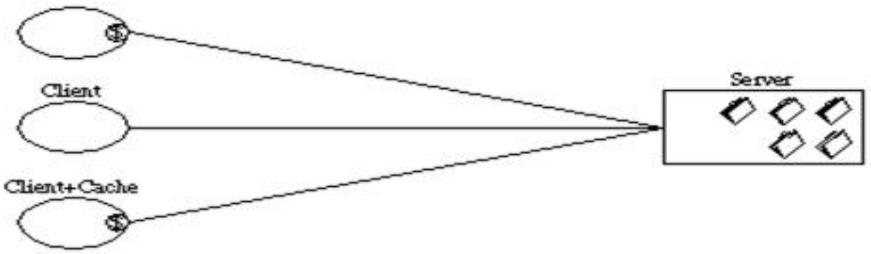

Figura 13 - Cliente - Cache - Server

IV.4 Sem Estado

Nesta arquitetura o servidor não armazena nenhuma informação de contexto. Toda informação necessária para atender a uma requisição deve estar contida nela mesma. Isto torna o servidor mais simples, pois ele não precisa levar em consideração o contexto atual para tomar decisões, toda informação necessária será enviada a ele a cada requisição. (FIELDING,2000)

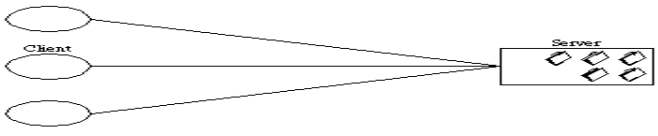

Figura 14 - Cliente - Sem estado - Servidor

V ARQUITETURA ORIENTADA A RECURSOS

V.1 Conceito

A arquitetura orientada a recursos (ROA) permite transformar um problema em um serviço web REST. Foi desenvolvida por Sam Ruby e Leonard Richardson.

ROA é composta por quatro conceitos:

- Recursos
- URIs
- Representações
- Links entre elas

e quatro propriedades:

- Endereçamento
- Falta de Estado
- Encadeamento
- Interface Uniforme

V.2 Recursos

Um recurso é algo que pode ser armazenado em um computador e representado por uma sequência de bits, por exemplo, um documento, uma imagem, um registro de uma tabela ou o resultado da execução de um algoritmo.(RICHARDSON,2007)

Exemplos de recursos disponíveis na web:

- Informações sobre REST
- Um mapa rodoviário
- Última versão de um determinado software
- A relação entre dois conhecidos: João e Maria
- Um diretório de recursos sobre Web Services
- Número de Vendas para o Q2-2012

V.3 URIs

O que torna um recurso um recurso? Ele tem que ter pelo menos uma URI. A URI representa o nome e o endereço de um recurso. Se uma parte das informações não tiver uma URI, não será um recurso e , portanto não estará na web. .(RICHARDSON,2007)

A URI é a tecnologia fundamental da web. Existiam sistemas de hipertexto antes do HTTP, mas eles não se comunicavam. A URI interconectou todos estes protocolos da INTERNET em uma web, da mesma forma como o TCP/IP interconectou redes como Usenet, Bitnet e CompuServe em uma única INTERNET.

Hoje navegamos na web, fazemos download de arquivos na web(não em sites FTP), pesquisamos publicações na web (não no WAIS) e conversamos na web (não nos

newsgroups Usenet). Os sistemas de controle de versão, por exemplo, o Subversion e o Arch, funcionam na web.

A web acaba com os outros protocolos, pois tem algo que a maioria dos protocolos não tem: um modo simples de identificar todo ítem disponível na web. Todo recurso na web tem pelo menos uma URI. Você pode obter uma URI a partir de qualquer browser e essa mesma URI pode ser enviada para um amigo, que o mesmo poderá acessar as mesmas informações. Pode parecer simples essa tarefa em nossos dias, mas esta interação seria impossível antes da existência das URIs.

V.4 URIs devem ser descritivas

As URI e seus recursos devem possuir uma correspondência clara. Abaixo estão algumas URIs ideais para os recursos listados acima:

- http://www.exemplo.com/wiki/rest;
- http://www.exemplo.com/mapa/rodoviario/nome_do_mapa;
- http://www.exemplo.com/software/versao/1.0.3.tar.gz;
- http://www.exemplo.com/relacionamento/joao;maria;
- http:/www.exemplo.com/search/webservices;
- http://www.exemplo.com/vendas/2012/02;

As URIS devem ser intuitivas para o usuário, de modo que os mesmos possam lê-las e interpretá-las com facilidade. (UNIFORM, 2011)

V.5 URIs x Recursos

Dois recursos podem ser iguais? Duas URIs podem designar o mesmo recurso? Uma única URI pode designar dois recursos?

Por definição dois recursos não podem ser iguais. Se fossem, teríamos apenas um recurso. Mas em algum momento dois recursos podem apontar para os mesmos dados. Se a versão atual de um determinado software for 1.03, então as URIs abaixo poderão representar os mesmos dados por um determinado período de tempo. Porém são dois conceitos distintos, um refere-se a uma determinada versão e o outro refere-se à versão mais recente. Até que surja uma nova versão, ambas as URIs estarão apontando para os mesmos dados.

- http://www.exemplo.com/software/versao/ultimaversao.tar.gz
- http://www.exemplo.com/software/versao/1.03.tar.gz;

Um recurso pode ter uma ou várias URIs. Se um recurso tiver diversas URIs , será mais fácil para os clientes referirem-se a ele. Em contrapartida alguns clientes podem utilizar uma URI ou outra perdendo-se assim o controle automático de qual URI está sendo utilizada.

Toda URI está associada a apenas um recurso. Porém, quando você recupera uma URI, o servidor pode enviar-lhe informações sobre diversos recursos: o solicitado e o relacionado a outros. Ao realizar uma pesquisa em um site de busca, o servidor disponibiliza além do recurso que estamos pesquisando, todas as informações que estão associadas a ele.

V.6 Endereçamento

O recurso é endereçável quando um usuário pode acessar os seus dados através de uma determinada URI.

O endereçamento é o aspecto mais importante de qualquer site ou aplicação web. Senão fossem pelo endereçamento os sites mais conhecidos, aqueles que utilizamos diariamente, não

seriam facilmente encontrados. Por exemplo, se quiséssemos encontrar informações sobre um determinado assunto, bastaríamos digitar no browser a URI abaixo:

- http://www.google.com.br/search?q=”assuntodesejado”;

Se o HTTP não fosse endereçável ou se o mecanismo de pesquisa do google não fosse uma aplicação endereçável, não seriamos capazes de utilizar a URI acima. Teríamos que realizar uma solicitação ao site do google e em seguida informar o objeto da pesquisa.

O endereçamento propicia a utilização de cache de requisições, ou seja, sempre que duas requisições forem realizadas para o mesmo recurso, em vez de buscar essas informações diretamente no servidor, pode-se utilizar um resultado já salvo da requisição anterior num proxy HTTP da rede local, assim evita o desperdício de banda . Isto é possível graças ao endereçamento que permite saber quais recursos já foram solicitados.

V.7 Falta de estado

Isso significa que toda solicitação HTTP ocorre em um isolamento completo. Quando o cliente faz uma solicitação HTTP, ele inclui as informações necessárias para que o servidor possa responder a requisição. (RICHARDSON,2007)

O servidor nunca guarda as informações das solicitações anteriores. Se elas fossem importantes, o cliente não teria que as enviar novamente.

O endereçamento diz que toda parte interessante da informação deve ser exibida como um recurso em sua própria URI. A falta de estado diz que os possíveis estados do servidor também são recursos e devem ser dados em sua própria URI.

Ao utilizar uma compra pela web, muitas vezes nos deparamos em situações onde o botão voltar do navegador não funciona da forma que esperamos. Isso acontece em decorrência do site violar o princípio da falta de estado. Tal site espera que você faça solicitações em uma ordem: A , B, C. Então ele fica confuso quando você faz a solicitação de B uma segunda vez, ao invés de ir para a solicitação C.

Um mecanismo de pesquisa é um serviço web com um número infinito de possíveis estados. Cada estado tem sua própria URI. Você pode solicitar ao serviço um diretório de recursos sobre REST: http://www.google.com/search?q=REST. Pode solicitar um diretório de recursos sobre SOAP: http://www.google.com/search?q=SOAP.

Quando você solicita um diretório de recursos sobre REST ou SOAP, apenas parte do diretório será recuperada em uma única página contendo uma lista dos 10 ou mais itens que o mecanismo de busca considera como os melhores resultados para a pesquisa. Para obter mais informações sobre o diretório, você deve realizar uma nova solicitação HTTP. A segunda página e as subsequentes são estados distintos da aplicação e possuem suas próprias URIs. Para recuperar uma segunda página sobre SOAP, teríamos que utilizar a URI: http://www.google.com/search?q=SOAP&start=10.

Em uma aplicação sem estado o cliente faz uma solicitação e termina onde começou. Cada solicitação é totalmente desconectada das outras. O cliente pode fazer solicitações para esses recursos várias vezes em qualquer ordem. Pode solicitar uma página2 com informações sobre SOAP antes de uma página1 com informações sobre REST e o servidor não se importará.

A figura 15 demonstra um mecanismo de pesquisa sem estado.

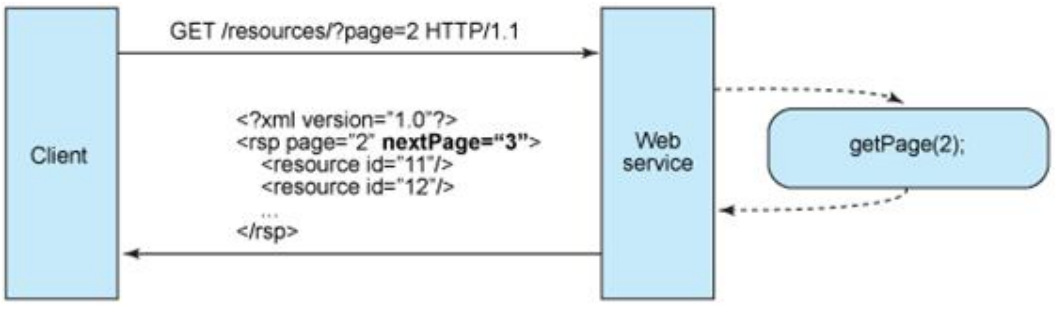

Figura 15 - Mecanismo de Pesquisa sem Estado

Em uma aplicação com estado, o cliente deve seguir uma ordem. Para chegar à página 2, ele necessariamente deve passar pela página 1.

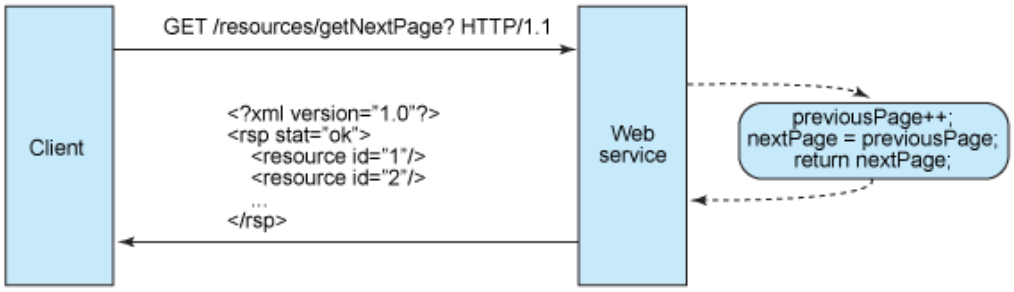

Figura 16 - Mecanismo de Pesquisa com Estado

Uma das vantagens da aplicação sem estado é que você é capaz de recuperar um diretório sobre REST até uma determinada página, por exemplo, a página 10 do diretório e voltar outras vezes na última página, objeto de sua pesquisa sem ter que passar pelas páginas anteriores, conforme URI:

- ✓ http://www.google.com/search?q=rest&start=10

V.8 Representações

Quando o servidor web envia uma informação de um recurso, esta informação é o resultado de uma série de bytes em uma linguagem específica, ou seja, não é possível enviar um recurso pela web, o que é transportado é uma representação do recurso. (RICHARDSON,2007)

Uma representação é composta de alguns dados sobre o estado atual de um determinado recurso. Um recurso é apenas um conceito abstrato de algo real, já a representação são os itens de dados que informam algo sobre o recurso. O número de vendas do último semestre de uma empresa poderia ser representado numericamente ou em um gráfico. Ambas as representações informam algo sobre o mesmo recurso.

As livrarias online costumam fornecer duas representações para seus clientes de um mesmo livro:

- ✓ Uma contendo apenas metadados, como uma imagem e as críticas do livro, usadas para fazer propaganda do mesmo.
- ✓ Uma cópia eletrônica dos dados do livro, disponibilizada para download mediante pagamento.

V.9 Múltiplas Representações

Se um recurso pode ter uma ou mais representações, como o servidor pode identificar qual representação do recurso, o cliente está solicitando?

Uma solução simples adotada pela ROA seria a utilização de uma URI distinta para cada representação de um recurso. Por exemplo, uma notícia publicada no idioma inglês e espanhol poderia possuir as seguintes URIs:

- ✓ http://www.exemplo.com/versao/1.04.em
- ✓ http://www.exemplo.com/versao/1.04.es

A desvantagem dessa prática é que você exibiria várias URIs para o mesmo recurso, dando a impressão que as várias URIs estão falando de coisas distintas. Para amenizar esse problema poderia ser criado uma URI "ideal", independente da língua, conforme abaixo:

- ✓ http://www.exemplo.com/versao/1.04

Esse modo alternativo é chamado de negociação de conteúdo, onde apenas a URI "ideal" é exibida. Quando um cliente faz uma solicitação para esta URI, fornece cabeçalhos de solicitação HTTP especiais que indicam qual tipo de representação o cliente deseja acessar.

Isso é possível, pois os navegadores web tem uma definição de preferência de língua, onde pode se optar pelo idioma que a representação deve ser exibida.

O mecanismo de pesquisa Google é uma boa opção para obter o resultado de uma pesquisa em praticamente todos os idiomas. Para isso basta apenas alterar a variável de definição h1na

URI (por exemplo, h1=tr para a língua turca). O mecanismo de pesquisa suporta a negociação do conteúdo de um recurso que possui várias representações.

V.10 Encadeamento

Quando realizamos uma pesquisa num site de busca, o servidor disponibiliza uma série de representações para o assunto pesquisado. Algumas representações possuem, além dos dados, links com outros recursos. Essas representações são conhecidas por hipermídia.

Peguemos novamente a URI:

- ✓ http://www.google.com/search?q=rest

Além do diretório de documentos do Google sobre REST, a página recupera um conjunto de links internos com outras páginas do diretório. A figura abaixo demonstra um encadeamento resultante da pesquisa.

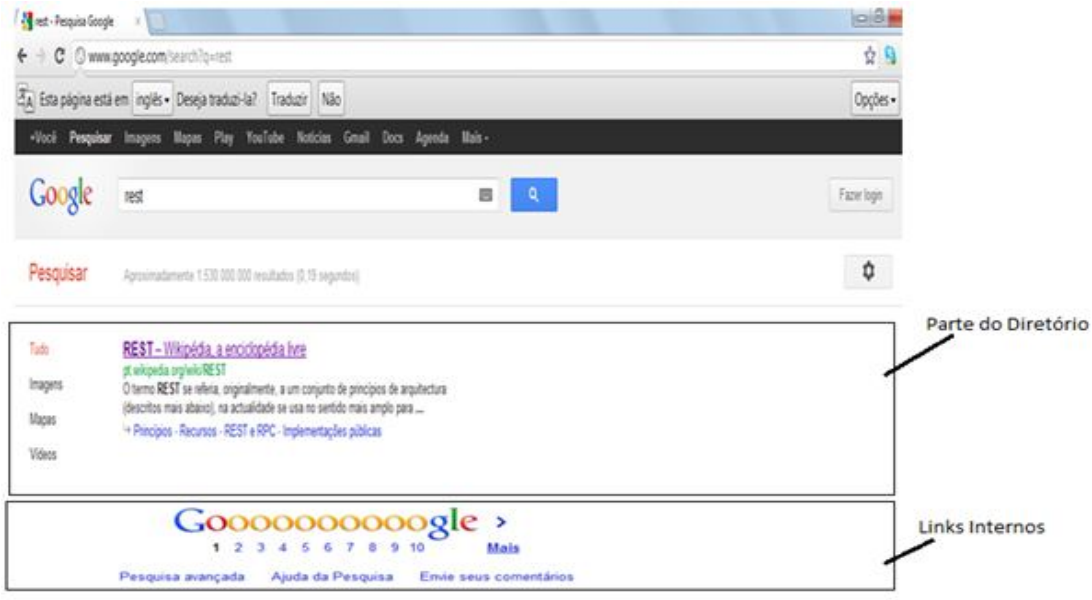

Figura 17 - Resultado da Pesquisa

Na figura temos dados e ligações. Os dados dizem que em algum lugar na web foi encontrado um diretório e links internos que fornecem acesso para outros recursos que informam algo sobre o assunto pesquisado: REST.

V.11 A Interface Uniform

A Interface Uniform define as operações possíveis para um recurso. O HTTP fornece as seguintes operações: (HYPERTEXT, 2011)

- ✓ GET: Recupera uma representação de um recurso.
- ✓ PUT: Cria ou atualiza um recurso.
- ✓ DELETE: Apaga um recurso existente.
- ✓ POST: Cria um recurso subordinado.
- ✓ HEAD: Recupera metadados de uma representação.
- ✓ OPTIONS: Verifica quais operações um determinado recurso suporta.

V.12 Segurança e Idempotência

Os métodos do HTTP podem ser classificados quanto a sua segurança e idempotência Uma solicitação é considerada segura, quando o método utilizado não altera o estado do recurso. Quando uma mesma solicitação ao ser executada várias vezes não altera o estado do recurso a denominamos idempotente.Todo método seguro é também idempotente. (RICHARDSON, 2007)

Os métodos GET, HEAD e OPTIONS são seguros e idempotentes, pois não alteram o estado do recurso.

PUT e Delete não são seguros, mas são idempotentes. O método POST é o único método que não é seguro nem idempotente.

VI REST X SOAP

Mencionarei abaixo as vantagens e desvantagens do SOAP em relação ao REST.

VI.1 Vantagens do SOAP

- ✓ Independência de transporte: Os cabeçalhos estão dentro da mensagem, ou seja, eles são independentes de protocolo para transportar a mensagem. O envelope SOAP pode ser transportado por qualquer protocolo: HTTP, SMTP, TCP, etc.
- ✓ Os conceitos sobre segurança são melhores especificados e difundidos nos protocolos baseados em SOAP do que no protocolo HTTP, utilizado pelo estilo REST.
- ✓ Os conceitos sobre transação são melhores especificados para o protocolo SOAP.

VI.2 Desvantagens do SOAP

- ✓ Não utilizam todo o "poder" do HTTP, limitando-se ao método POST para realizar múltiplas operações.
- ✓ É muito trabalhoso criar uma interface WSDL manualmente, apesar de terem ferramentas para gerá-las automaticamente, as mesmas tendem ser frágeis.
- ✓ Todo serviço está sob a mesma URI, logo não há encadeamento entre os recursos. Para resolver esse problema é necessário criar uma WADL, o qual descreve como um recurso se liga a outro.
- ✓ Assim como é trabalhoso criar uma WSDL, acontece o mesmo quando refere-se à criação do UDDI, talvez seja esse um dos motivos pelo quais ele não se popularizou.
- ✓ Não pode ser armazenado em cache, pois o SOAP usa o método HTTP POST, o que é considerado não seguro, como vimos esse método pode alterar os dados de um recurso.

VII Conclusão

É preciso ter cuidado ao apontar uma tecnologia como sendo superior a outra. Geralmente somos levados pelo modismo e seguimos a tendência, sem ter um embasamento que justifique a nossa escolha. O intuito desse trabalho foi demonstrar a funcionalidade das abordagens que são mais utilizadas nos serviços web. É claro que dentro do contexto em que estamos inseridos, uma tecnologia pode levar vantagens em relação a outra, conforme foi demonstrado no capítulo anterior.

Podemos concluir que a melhor escolha estará atrelada ao cenário em que estamos inseridos. Talvez a nossa escolha dentro de um contexto não seja a mais adequada em outro.

REFERÊNCIAS BIBLIOGRÁFICAS

RICHARDSON, S. R. L. RESTful Serviços Web. [S.l.]: O'Reilly Media, 2007.

FIELDING, R. Architectural Styles and the Design of Network-based Software Architectures. Tese (Doutorado) — UNIVERSITY OF CALIFORNIA, IRVINE, 2000.

GOMES, D. A. Web Services SOAP em Java: Novatec, 2010.

HYPERTEXT Transfer Protocol - HTTP/1.1 (RFC 2616 Fielding, et al). Disponível em: http://www.w3.org/Protocols/rfc2616/rfc2616.html. Acesso em: 14 set. 2011.

EXTENSIBLE Markup Language (XML). Disponível em: http://www.w3.org/XML/. Acesso em: 14 set. 2010.

UNIFORM Resource Identifier (URI): Generic Syntax. Disponível em: http://labs.apache.org/webarch/uri/rfc/rfc3986.html. Acesso em: 14 set . 2011.

SIMPLE Object Access Protocol (SOAP). Disponível em: http://www.w3schools.com/soap/.Acesso em 14 set. 2011.

WEB Services. Disponível em: http://www.w3schools.com/webservices/. Acesso em 14 set. 2011.

UNIVERSAL Description Discovery and Integration (UDDI). Disponível em: http://www.w3schools.com/uddi/. Acesso em 14 set. 2011.

REMOTE Procedure Call – RPC. Disponível em: http://saloon.inf.ufrgs.br/twiki-data/Disciplinas/INF01151/ExercicioEmLaboratorio2006-2-04/rpc.pdf. Acesso em 14 set. 2011