PROPOSTA DE UM CONJUNTO DE BOAS PRÁTICAS PARA A UTILIZAÇÃO E CONSTRUÇÃO DE SERVIÇOS WEB BASEADOS EM REST

Proposal for a set of good practice for the use and construction of Web service Rest based

ANDRADE, Guilherme Afonso
Faculdade Politécnica de Campinas

SILVA, Neimar Gustavo Lopes da
Faculdade Politécnica de Campinas

DIAS, Mateus Pereira Dias
Faculdade Politécnica de Campinas

Resumo*:* A fim de facilitar a implementação dos serviços web e utilizar ao máximo os recursos consagrados no HTTP, tem sido sugerido como alternativa o estilo arquitetural Representational State Transfer (REST). Neste artigo, serão apresentados os conceitos necessários para o desenvolvimento de uma arquitetura baseada no estilo REST e um conjunto de boas práticas para tornar a implementação dos serviços bem sucedida.

Palavras-chave*:* REST, Serviços Web, SOAP.

Abstract*:* In order to facilitate the implementation of web services and use the maximum of devoted resources in HTTP, it's been suggested as alternative the architectural style Representation State Transfer (REST). In this article, will be presented the necessary concepts to develop an architecture based on REST style and a set of good practices to turn services implementation successful.

Keywords*:* REST, Web Services, SOAP.

1. Introdução

A busca de soluções para integração de sistemas e comunicação entre diferentes aplicações incentivou a popularização dos serviços web, que são

definidos como sistemas de software com o objetivo de prover um serviço de interoperabilidade entre sistemas distintos, que funcionam, ou não, em mesma rede. [1].

A comunicação com os serviços web é possibilitada por meio da Internet em um estilo cliente/servidor e, sua utilização, objetiva a troca de informações entre aplicações. Este conceito pode ser chamado de web programável que é muito parecida com os sites convencionais que retornam páginas amigáveis e atraentes, com conteúdos que servem para o consumo humano. A web programável tem como principal objetivo disponibilizar representações para uso programático, geralmente por outras aplicações. [2].

Para a disseminação e a adoção dos serviços web surgiram padrões para prover integração entre diferentes aplicativos de software em diversas plataformas [3]. Esses modelos fornecem definições comuns de um serviço web, destacando-se, entre os principais: SOAP[1], WSDL[2], UDDI[3].

Apesar dos muitos padrões em torno dos serviços web, a maior parte deles negligenciou os conceitos implementados no protocolo atual da web (HTTP[4]). Este fato torna difícil a implementação dos serviços web.

Uma alternativa para implementação de serviços web, utilizando ao máximo os recursos que consagraram o HTTP, é o estilo arquitetural Representational State Transfer (REST), sugerido por Roy T. Fielding em sua pesquisa de doutorado [4], onde é apresentado um modelo para o desenvolvimento de sistemas de hipermídia distribuídos

Pretende-se demonstrar neste trabalho, que a construção e a utilização de serviços web baseados em REST é mais eficaz, e seu entendimento, é claro, diferenciado de outros padrões, como o SOAP que é de difícil compreensão e uso.

[1] Simple Object Access Protocol
[2] Web Services Definition Language

2. Objetivo

O objetivo deste trabalho é apresentar um conjunto de boas práticas para a utilização do estilo de arquitetura REST, visando à utilização fácil e eficaz da arquitetura para a implementação e utilização de serviços web.

3. Procedimentos metodológicos

Para a construção de um conjunto de boas práticas no uso de REST, foram realizadas as seguintes etapas:

- Levantamento teórico sobre o estilo de arquitetura REST;
- Análise do funcionamento de serviços web implementados utilizando tal tecnologia e seus resultados;
- Elaboração de um conjunto de boas práticas, baseado nos resultados obtidos na etapa anterior.

4. Justificativa

Existem vários artigos que explicam os conceitos e as definições de REST, segundo o modelo apresentado por Roy Fielding em [4]. Contudo, o pouco material que é encontrado contendo implementações de serviços baseado no modelo, não apresenta um conjunto de boas práticas, como é proposto por este trabalho.

5. O estilo arquitetural REST

O estilo arquitetural REST tem como princípio utilizar os próprios recursos existentes no protocolo HTTP como meio para prover serviços

[3] Universal Description, Discovery and Integration
[4] Hypertext Transfer Protocol

distribuídos. Tem como base as estruturas definidas no HTTP Object Model [8] e possui uma série de restrições que levam um sistema a possuir determinadas propriedades que não violam os princípios da WEB.

Uma afirmação sobre como o modelo REST se comporta com a WEB pode ser encontrada em [5] página 116: *"The name "Representational State Transfer" is intended to evoke an image of how a well-designed Web application behaves: a network of Web pages forms a virtual state machine, allowing a user to progress through the application by selecting a link or submitting a short data-entry form, with each action resulting in a transition to the next state of the application by transferring a representation of that state to the user."*. Em resumo, REST representa um modelo de como a Web deveria funcionar.

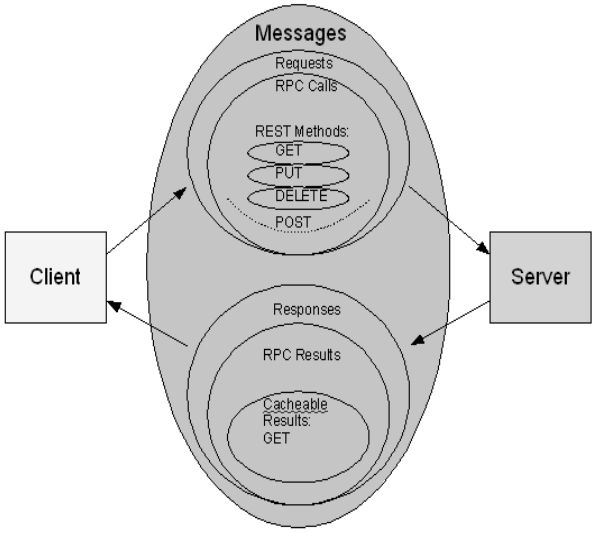

Figura 1 - Ciclo de comunicação REST [13]

A seguir serão apresentados os conceitos e restrições que uma arquitetura de serviços REST deverá atender:

a) Recursos

O recurso é um importante conceito dentro do modelo, caracterizando-se como uma abstração de um elemento de informação dentro do sistema, que pode representar qualquer objeto que possa ser nomeado, seja físico, seja

abstrato. Um recurso é um mapeamento conceitual para um conjunto de entidades [4]. Alguns exemplos de possíveis recursos são:

- Um documento;
- Uma imagem;
- A relação entre dois recursos.

Todo recurso deve possuir um nome e um endereço, representado por uma URI[5], que é uma cadeia de caracteres usada para identificar, de modo simples, todo item disponível na web [6]. Por pertencer a um namespace global, toda URI possui uma identificação única e universal [7].

É recomendável que toda URI possua uma estrutura previsível e descritiva do recurso que está representado. Não é uma regra absoluta, mas já é entendida como uma boa prática no desenvolvimento de aplicativos web, que facilita a sua leitura e compreensão por usuários comuns. Alguns exemplos de URI:

http://www.policamp.edu.br/alunos/<ra>

http://www.policamp.edu.br/materias/<id>/alunos

Todo recurso deve possuir um endereço intuitivo na obtenção dos dados de um recurso. Em outras palavras, o endereço deve significar o que o serviço faz.

b) Falta de Estado

Toda iteração HTTP deve ocorrer em um nível de isolamento completo, ou seja, quando é realizada uma requisição, esta deve possuir todas as informações necessárias para o processamento no servidor e este, por sua vez, não pode depender de informações de solicitações anteriores. Se alguma informação do estado anterior for importante para a nova solicitação, o cliente deve mencionar na URI esses dados a serem enviados para o servidor.

[5] Uniform Resource Identifier – entende-se como um endereço web

Por exemplo, ao realizar a solicitação de uma lista de objetos, esta não deve retornar todo seu conteúdo e sim, os cinco primeiros registros para facilitar a exibição. Na primeira requisição, é necessário realizar uma chamada como: http://www.policamp.edu.br/materias/<id>/arquivos. Para consultar os próximos registros, o cliente deve informar ao servidor o estado atual da aplicação, realizando uma solicitação para a URI: http://www.policamp.edu.br/materias/123/arquivos?start=5. Ou seja, o cliente "avisa" o seu estado ao servidor, informando que é necessário listar as representações apenas a partir do sexto registro. É uma solicitação sem estado, pois cada uma é desconectada das outras. O cliente pode fazer solicitações para páginas distintas, mesmo não seguindo uma ordem, que o servidor não se importará.

Para construir o estado no lado do servidor, poderiam ser utilizadas técnicas como sessões e cookies, porém a característica da arquitetura obriga que o estado do recurso seja mantido na máquina cliente e transmitido para o servidor em toda solicitação [2].

c) Representações

A representação é o formato específico enviado pelo serviço, cuja finalidade é transmitir a ideia do recurso, que pode conter qualquer informação útil sobre seu estado, sejam dados reais, sejam metadados.

d) Link e encadeamento

A utilização de hipermídia como mecanismo de aplicação do estado [4] é a restrição mais obscura e discutida no estilo REST [9]. Muitas vezes sua utilização é negligenciada e não implementada.

Algumas representações possuem dados que são extraídos e, depois, descartados. Mas muitos dos recursos possuem representações com documentos de hipermídia, isto é, não possuem apenas dados, mas links com outros recursos.

Com isso, o estado atual da aplicação não é armazenado no servidor como um estado do recurso, mas controlado pelo cliente como estado da aplicação. O estado é apenas dirigido pelo servidor, ao informar ao cliente os caminhos dos links, fornecendo à hipermídia.

Esta restrição ajuda na evolução independente dos sistemas, pois quando é alterada a estrutura de navegação do servidor, este deve fornecer as novas URIs em seus links, e o cliente continuará operando sem sofrer adaptações.

e) Interface Uniforme

Todos os métodos necessários para a comunicação com o servidor estão presentes na interface uniforme do HTTP e devem ser usados para lhe enviar o estado do recurso. Os métodos utilizados são:

- HTTP HEAD
- HTTP OPTIONS
- HTTP GET
- HTTP DELETE
- HTTP PUT
- HTTP POST

* HTTP HEAD

O método HEAD é usado para obter os detalhes dos metadados do recurso, sem realizar o download dos dados da entidade. Ele é útil para verificar a existência do recurso ou descobrir detalhes das informações do escopo do serviço.

* HTTP OPTIONS

O método OPTIONS permite descobrir as operações que são permitidas para um determinado recurso solicitado, transmitidas pelo servidor por meio de uma resposta com um cabeçalho de permissões: *Allow*. Por exemplo:

Allow: GET, HEAD, PUT, DELETE

* HTTP GET

O método GET permite recuperar as informações de um recurso, retornando este os metadados no cabeçalho de resposta e a representação no corpo da entidade.

* HTTP DELETE

O método DELETE permite apagar as representações de um recurso, enviando uma solicitação para sua URI. Esta operação deve retornar o status HTTP do resultado da ação.

* HTTP PUT

O método PUT permite realizar a modificação ou a criação de um recurso. O corpo da entidade de solicitação deve conter os dados que serão enviados para o servidor.

A alteração desses dados ocorre quando é realizada uma solicitação para um recurso já existente, como, por exemplo, a URI http://www.policamp.edu.br/alunos/<RA>. Neste caso, o registro do aluno será alterado com dados que estão no corpo da entidade.

Para a criação um novo recurso com um PUT, deve ser realizada uma solicitação para uma URI não existente, e o endereço solicitado será o caminho para o novo recurso. Esta ação obriga o conhecimento prévio do endereço do novo recurso.

* HTTP POST

O POST é o mais complexo dos métodos existentes na interface HTTP [2]. Maiores detalhes sobre este método podem ser encontrados em seu texto original, o RFC 2616 [10], entretanto serão apresentadas apenas as finalidades que se encaixam no estilo REST. A ação real que o servidor irá executar sobre uma solicitação POST depende da URI solicitada. Pode ser utilizada para criar

um novo recurso ou simplesmente anexar dados ao estado de um recurso já existente.

Partindo das perspectivas do REST, pode ser usado para criar um recurso que seja subordinado a outro, chamado recurso "pai". Isto se torna útil quando não se conhece a nova URI, pois depende de dados processados no servidor. Neste caso, o cliente precisa apenas ter ciência da URI do recurso pai. Também é possível utilizar uma solicitação POST para anexar as informações enviadas para o estado de um recurso já existente. Neste caso, não é criado um novo recurso, e sim a adição dos dados. Além das utilizações apresentadas, é possível realizar uma implementação que foge dos conceitos sugeridos para estilo REST, o chamado POST sobrecarregado, quando utiliza apenas um método HTTP para representar qualquer quantidade de métodos não HTTP. Esta implementação não é recomendada, pois a informação do método não se encontra na interface HTTP.

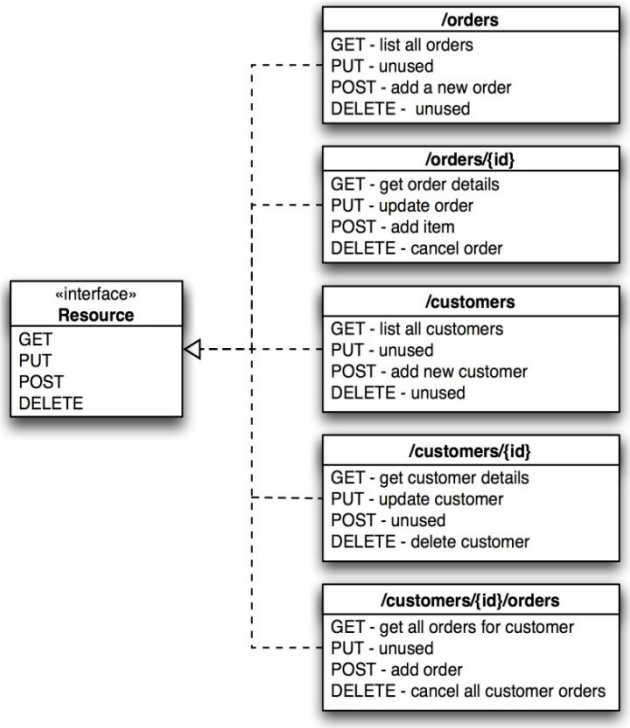

Figura 2 - Exemplos de Recursos e o retorno de cada um de seus métodos [7]

6. Contribuição: conjunto de boas práticas dos serviços baseados em REST

As restrições já apresentadas são peças fundamentais para uma base sólida nos princípios do estilo. Algumas práticas melhores serão indicadas para tornar a implementação dos serviços bem sucedida. Este conjunto de práticas foi possível pelo trabalho experimental em laboratório.

a) Nomeação dos recursos

Os nomes dos recursos devem ser divididos de forma a conter e a suportar todas as partes móveis de que o sistema tratará [2]. A divisão será realizada pelos substantivos [11] que compõem o conjunto de dados desse sistema, como, por exemplo, “Aluno”, “Matéria” e “Curso”, que representam um recurso dentro da aplicação; para acessá-los, devem possuir um endereço intuitivo, claro e descritivo sobre o conjunto de dados que retornará ao cliente, contendo todas as informações necessárias ao escopo do serviço.

Quando for necessário representar outros níveis dentro de um determinado recurso, usando a barra para separar as partes das informações do escopo, devem ser utilizadas hierarquias, que representam os vários níveis da informação dentro de um diretório [2]. Para representar as notas ou matérias de uma determinada classe de alunos, podem ser utilizados níveis hierárquicos dentro do recurso desse aluno, como nos exemplos a seguir:

http://www.policamp.edu.br/alunos/notas

http://www.policamp.edu.br/alunos/materias

b) Construção dos dados aceitos pelo serviço

Em muitos casos, é necessário o envio de informações do estado para o serviço no momento de uma chamada. É desejável que todas as informações do escopo sejam enviadas ao endereço do serviço, garantindo o princípio de endereçabilidade do REST [4]. Quando for necessário o envio de uma representação em conteúdo binário, pode se utilizar o cabeçalho para este fim.

Quando são conhecidos os valores a serem transmitidos no escopo, devem ser usadas variáveis de diretório dentro dos endereços dos recursos [2]. Como exemplo para esta abordagem, a consulta de um determinado aluno, dentro do diretório de alunos: www.policamp.edu.br/alunos/<RA>. É possível notar como se fosse uma relação todo parte, ou seja, um aluno pertence a um conjunto maior: alunos.

Outra abordagem possível é a utilização de variáveis de consulta, para enviar parâmetros ao servidor em recursos algorítmicos. Uma consulta dentro do acervo de livros de uma biblioteca pode ser implementada utilizando este conceito: www.policamp.edu.br/biblioteca/livro?filtro=<parâmetros do filtro>.

c) Representações retornadas pelo serviço

Os serviços podem retornar uma variedade de tipos de dados que devem conter todas as informações úteis sobre o recurso. É importante que os dados de retorno satisfaçam os objetivos esperados pelo cliente para poder ser utilizado adequadamente.

Devido à facilidade de implementação e de integração entre diferentes linguagens de programação, a utilização do formato XML[6] para retorno dos dados é um padrão adotado. Se algum outro formato for mais útil ao consumidor do serviço, deve ser usado, pois não infringirá os princípios REST.

No conteúdo de retorno, é imprescindível fornecer os links para os recursos vizinhos da representação, os quais são os possíveis novos estados da aplicação, tendo como objetivo promover o encadeamento entre serviços, o que corresponde à capacidade de ir de um recurso para outro, utilizando esses links.

Uma representação em XML de um aluno com links para outros recursos:

[6] eXtensible Markup Language

```
<?xml version=”1.0” standalone=”Yes”>
<aluno>
    <nome>...</nome>
    ...
    <curso href=“/cursos/SI”/>
    <notas href=“/alunos/<RA>/notas”/>
</aluno>
```

d) Tratamento das condições

Os serviços de leitura e gravação devem tratar de todas as condições de sucesso e falha para a chamada dos recursos, importantes para assegurar que as solicitações do cliente sejam transformadas corretamente em respostas. O tratamento do status de retorno deve utilizar os códigos de respostas do HTTP.

O status padrão de sucesso para os serviços de leitura e gravação, é o código de retorno 200 (“OK”), com as representações no corpo da entidade e possíveis dados no cabeçalho HTTP.

Na criação de um novo recurso, sugere-se retornar ao status 201 (“Criado”), com um cabeçalho de resposta *Location*, com a URI do novo recurso.

Quando for solicitado um recurso inexistente dentro do sistema, deve-se retornar a um código de status 404 (“Não Encontrado”), não sendo necessário enviar o corpo da entidade na resposta.

Se forem informados valores inválidos para os parâmetros do serviço, ou a obrigatoriedade de parâmetros, deve ser indicado o status 400 (“Requisição Incorreta”). É adequado enviar no corpo da solicitação, informações sobre os valores incorretos para um possível ajuste.

Para serviços que implementam autenticação e forem solicitados com credenciais incorretas, o sistema deve retornar ao código de resposta 401 (“Não autorizado”).

Caso o serviço não seja capaz de satisfazer a solicitação devido à sobrecarga no servidor, deve ser enviado o status 503 (“Serviço Indisponível”). Todos os status de erro pertencentes à família 5xx representam erros ocorridos no servidor, quando o cliente fica incapacitado de obter as representações do serviço.

Outros problemas podem ocorrer durante o consumo de um serviço, porém os erros apresentados acima são suficientes para tratar da grande parte dos casos típicos.

e) Segurança

Toda solicitação GET e HEAD deve ser segura, não alterando o estado do servidor [12]. Se o software cliente invocar o recurso uma ou N vezes, o seu estado deve ser o mesmo, como se nunca existisse a solicitação. Este conceito é importante por se tratar de métodos confiáveis dentro do protocolo HTTP. Quando solicitado um recurso com estes métodos e o servidor não retornar uma resposta, é seguro fazer uma nova solicitação.

f) “Idempotência”

As operações “idempotentes” são aquelas que têm o mesmo efeito sempre que são executadas uma ou mais vezes. Os métodos GET, HEAD, PUT e DELETE devem ser idempotentes [12]. Se um recurso for alterado com PUT, o estado deste será alterado no servidor. Realizando novas solicitações PUT para o mesmo recurso, a representação do estado será igual à da primeira solicitação realizada.

g) GET Condicional

A utilização de GET condicional permite que o cliente e o servidor trabalhem juntos para economizar largura de banda. Quando o cliente executa

uma consulta a um determinado recurso e o servidor verifica que o recurso do cliente é exatamente igual ao do servidor, ele retorna apenas uma resposta ao agente do usuário, informando que esse recurso é o mesmo e que o cliente pode se utilizar de seu cache para devolver o recurso ao usuário, economizando assim, na transferência de dados.

h) Compactação

Outro mecanismo utilizado para a economia de dados na rede é a compactação de dados. Um cliente HTTP pode requisitar uma versão compactada das representações e a descompactação ocorre de modo transparente. Para requisitar dados compactados, o cliente envia um cabeçalho *Accept-Enconding*, informando os tipos de compactação compreendidos. Se o servidor entender algum dos formatos enviados, ele pode transmitir os dados compactos, economizando largura de banda. O cabeçalho *Content-Encoding* deve ser enviado na resposta, indicando o padrão de compactação do documento. Exemplo de uma solicitação e resposta com compactação de dados:

Solicitação:

GET /aluno/<RA> HTTP/1.1

Host: www.policamp.edu.br

Accept-Encoding: gzip, compress

Resposta:

200 OK

Content-Type: text/XML\

Content-Enconding: gzip

7. Conclusões

A Transferência de Estado Representacional (REST) é uma tecnologia promissora para o desenvolvimento de sistemas web distribuídos. O estilo possui uma fácil implementação e funciona de modo a aproveitar todos os recursos já existentes no protocolo HTTP, possibilitando, com isso, a utilização de uma forma simples nos mais diversos tipos de equipamentos que suporte o HTTP – protocolo padrão da Internet.

Outro ponto importante do modelo é a quantidade de dados trafegados na rede. Por não implementar protocolos adicionais como o SOAP, durante o transporte da solicitação e da resposta, são trafegados apenas os dados que agregam valor para o serviço. Técnicas de compactação e de resposta condicional são facilmente implementadas por serem nativas do protocolo HTTP. Esta abordagem é muito eficiente para clientes que possuem um alto custo de consumo de banda, como celulares e serviços com grande número de acessos, evitando, assim, a transferência de dados que não foram solicitados pelo software cliente. Mesmo para os problemas que os serviços web tradicionais tentam resolver, como aplicações de negócios e governamentais, a utilização de serviços REST mostra-se eficiente, tornando uma alternativa real para a implementação dos serviços. Com a utilização dos modelos apresentados, todo o esforço é na implementação do recurso, não nos protocolos e regras adicionais impostos pelos serviços tradicionais.

Referências

[1] W3C, *Web Services Glossary*, Fevereiro 2004. [Online]. Disponível em: http://www.w3.org/TR/ws-gloss. Último acesso em: 28/11/2009.

[2] L. Richardson and S. Ruby, RESTful Serviços Web. Rio de Janeiro: Alta Books, 2000.

[3] W3C, *Web Services Architecture*, Fevereiro 2004. [Online]. Disponível em: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211. Último acesso em: 30/11/2009.

[4] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doctor of Philosophy in Information and Computer Science, Chapter 5) - University of California, Irvine, 2000.

[5] R. T. Fielding and R. N. Taylor, *Principle Design of the Modern Web Architecture*. ACM Transactions on Internet Technology, pp 115–150, Maio 2002.

[6] W3C, *Universal Resource Identifiers - Axioms of Web Architecture*, Dezembro 1996. [Online]. Disponível em: http://www.w3.org/DesignIssues/Axioms.html. Último acesso em: 30/11/2009.

[7] S. Tilkov, *"A Brief Introduction to REST"*, Dezembro 2007. [Online]. Disponível em: http://www.infoq.com/articles/rest-introduction. Último acesso em: 30/11/2009.

[8] S. Nunes and G. David, *"Uma Arquitectura Web para Serviços Web"*. [Online]. Disponível em: http://www.fe.up.pt/si/publs_web.show_publ_file?p_id=12085&pv_nocache=20091129234207. Último acesso em: 28/11/2009.

[9] *REST intro slides*. Maio 2007. [Online]. Disponível: http://tech.groups.yahoo.com/group/rest-discuss/message/8367?var=1. Último acesso em: 30/11/2009.

[10] Network Working Group, *Hypertext Transfer Protocol - HTTP/1.1*, Junho 1999. [Online]. Disponível: http://www.ietf.org/rfc/rfc2616.txt. Último acesso em: 30/11/2009.

[11] R. L. Costello, "*Building Web Services the REST Way*". [Online]. Disponível em: http://www.xfront.com/REST-Web-Services.html. Último acesso em: 30/11/2009.

[12] D. Gourley and B. Totty, *HTTP: The Definitive Guide*. O'Reilly Media, 2009.

[13]http://www.innoq.com/blog/st/2006/06/30/rest_vs_soap_oh_no_not_again.html Ultimo acesso em: 17/12/09