Web services WS-* versus Web Services REST

Tharcis Dal Moro, Carina F. Dorneles, Marcelo Trindade Rebonatto

Instituto de Ciências Exatas e Geociências – Universidade de Passo Fundo (UPF)
Passo Fundo – RS – Brasil

tharcis@gmail.com,dorneles@inf.ufsc.br,rebonatto@upf.br

Resumo. *Este artigo apresenta um estudo sobre Web services utilizando a arquitetura WS-* e a arquitetura ROA (RESTful), bem como um detalhamento das principais tecnologias envolvidas. Este artigo também apresenta um estudo de caso desenvolvido na plataforma Java a fim de demonstrar o funcionamento e principais diferenças em se trabalhar com Web services que utilizam as duas metodologias citadas.*

Abstract. *This paper presents a study about Web services using the WS-* architecture and the ROA architecture (RESTful), as well as some details of the main technologies involved. This paper also presents a case study developed on Java platform for the purpose of demonstrate the functionality and main differences in working with Web services using WS-* and ROA.*

1 Introdução

A necessidade da integração entre aplicações, a utilização unificada de processos encontrados em diferentes sistemas e escritos em diferentes linguagens são alguns exemplos de carências encontradas em empresas nos dias de hoje. A fim de sanar estas questões, a tecnologia dos *Web services* foi criada, permitindo assim, disponibilizar formas de integrar sistemas distintos, modularizar serviços e capacitar a integração e consumo de informações [Menéndez 2002].

Empresas e corporações empregam abordagens de *Web services* dependendo de suas necessidades. A escolha de se utilizar SOAP, RPC, HTTP puro (*RESTful*) deve ser realizada com base em estudos e comparações para que a melhor solução seja aplicada. Como cada abordagem possui suas vantagens e desvantagens, deve-se levar em conta a opção que melhor atende os requisitos da aplicação.

Problemas complexos exigem soluções mais bem elaboradas. A utilização de meios complexos para solucionar questões triviais, além de trazer gastos desnecessários às empresas, se torna lento e custoso em termos de infraestrutura. Foi com o intuito de esclarecer o real potencial e aplicabilidade das abordagens WS-*[1] e REST que a elaboração deste trabalho foi impulsionada.

Este artigo apresenta um estudo sobre *Web services* utilizando a arquitetura WS-* e *Web services* que seguem o estilo de arquitetura REST (ROA), bem como um estudo de caso para demonstrar as duas abordagens e servir de instrumento no processo de comparação entre ambas. A principal contribuição do estudo realizado é

[1] O termo WS-* foi definido neste artigo para indicar Web services que utilizam SOAP e demais especificações como WSDL e UDDIs

proporcionar um embasamento macro da tecnologia e destacar as principais características das arquiteturas em questão.

O artigo está dividido como segue. A Seção 2 apresenta alguns trabalhos relacionados a este artigo. A Seção 3 introduz a tecnologia de *Web Services*, mostrando conceitos importantes das tecnologias WS-* e REST. A Seção 4 apresenta um estudo de caso de serviços utilizando a plataforma Java, seguido de uma análise comparativa na Seção 5. As conclusões e os trabalhos futuros são descritos na Seção 6.

2 Trabalhos relacionados

A maioria dos trabalhos na literatura, que compara características de *Web services* REST e WS-*, normalmente foca em aspectos técnicos teóricos [Newmarch 2005; Shi 2006; Pautasso 2007; Chinthaka 2007; Pautasso 2009]. O trabalho descrito em [Chinthaka 2007] apresenta uma comparação teórica entre *Web services* REST e WS-*, focando em como REST pode ser utilizado com a WSDL 2.0. Já o trabalho descrito em [Prescod 2002] apresenta métodos para padronizar os protocolos usados para transferir documentos XML que procuram minimizar as fraquezas e maximizar pontos fortes de cada protocolo. O trabalho descreve os métodos de forma teórica, sem experimentos que possam validar as idéias apresentadas.

Em [Newmarch 2005], o autor enfatizou o uso de ambas as abordagens na arquitetura UPnP, o qual é um *middleware* projetado para rede *plug and play*. O trabalho dá ênfase a pontos negativos do SOAP, já apresentados em outros trabalhos, e mostra como estes pontos negativos se refletem também em ambientes UPnP. O autor mostra como mecanismos construídos em princípios REST podem produzir sistemas UPnP menores, mais rápidos e mais simples.

A abordagem apresentada em [Shi 2006] foca em *Web services* semânticos. Combina as melhores características provenientes de *Web services* SOAP e REST e explora o que eles têm em comum. Apresenta um estudo que relaciona os *Web services* e a Web semântica, ambas sobre HTTP e sem a utilização de *frameworks* extra, tais como SOAP ou WSDL. Segundo o autor, este método permite aos usuários utilizar e compartilhar serviços fazendo uso de computação distribuída.

Um estudo de caso, na área de *cross-organization workflow*, comparando REST e WS-* é apresentado em [MUEHLEN et. al 2005]. O trabalho apresenta uma extensa comparação teórica entre as duas abordagens, apontando vantagens e desvantagens em cada um dos casos. O estudo de caso é feito com serviços baseados em SOAP, com projetos estritamente similares às chamadas de procedimento remoto, e os baseados em REST, com projetos de baixo acoplamento, semelhante à navegação de links da web. Os testes e a análise final foram definidos e observados sob os pontos de vista de clientes e vendedores de uma aplicação Web.

O que se observa nos trabalhos da literatura é que grande parte deles apresenta relatos teóricos quanto a vantagens de um ou de outro formato de *Web services*, sem apresentar dados conclusivos reais sobre o uso prático de cada um deles. Desta forma, o ponto fraco dos trabalhos existentes, consiste na falta de relatos que mostrem comparações efetuadas durante o uso de ambas as arquiteturas para *Web services*. O presente trabalho se distingue dos demais por apresentar um relato de experimentos executados com um estudo de caso projetado em ambas as arquiteturas, a fim de que se pudesse comparar questões práticas de uso.

3 Conceitos Básicos

Um *Web Service* é um sistema de software desenvolvido para suportar interoperabilidade entre máquinas sobre uma rede, o qual pode apresentar uma interface que o descreve (WSDL, WADL). Outros sistemas interagem com os *Web services* por meio de mensagens SOAP, geralmente usando HTTP com serialização XML em conjunto com outros padrões relacionados a Web [W3C 2004].

Com esta tecnologia, torna-se possível que aplicações diferentes interajam entre si e sistemas desenvolvidos em plataformas diferentes se tornem compatíveis. Os *Web services* são componentes que permitem que aplicações enviem e recebam dados em formatos variados. Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal, como é o caso do formato XML.

Uma característica fundamental dos *Web services*, diz respeito à possibilidade de utilização de diferentes formas de transmissão de dados pela rede. Logo, a arquitetura de *Web services* pode trabalhar com protocolos, tais como HTTP, SMTP, FTP, RMI/IIOP ou protocolos de mensagem proprietários [W3C 2004].

As primeiras versões das especificações de *Web services* ofereciam apenas o HTTP como meio de transporte de dados (mensagens SOAP XML) e comunicação entre clientes e serviços, sendo ainda o mais utilizado atualmente.

*3.1 Web Services WS-**

No final da década de 90, milhares de pessoas estavam desenvolvendo sistemas de *e-commerce* baseados em HTTP e XML. Cada qual com sua própria maneira de implementar segurança, confiabilidade, gerenciamento de transações. Com isso, um caos começou a ser formado: incompatibilidades entre sistemas, dificuldade na manutenção e em contrapartida, a necessidade de se criar uma maneira comum de realizar estas tarefas. Estes fatos impulsionaram o surgimento dos padrões WS-*. Hoje mantidos pela W3C e OASIS [Weerawarana, *et al*. 2007].

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

3.1.1 SOAP (*Simple Object Access Protocol*)

Nos padrões WS-*, as mensagens trocadas entre serviço e cliente consumidor devem ser armazenadas em envelopes SOAP. Este protocolo de comunicação dita um formato de envio de mensagens entre aplicações, o qual é descentralizado e distribuído, ou seja, qualquer plataforma de comunicação pode ser utilizada, seja ela proprietária ou não [W3C 2000].

Segundo Streibel (2005), este protocolo define uma estrutura para interoperabilidade entre sistemas através de mensagens XML. Um envelope SOAP é um documento XML, o formato deste documento é definido por um XML *schema*, que, por sua vez, faz uso de XML *namespaces* para garantir extensibilidade. Um documento SOAP é

basicamente constituído de um elemento raiz *Envelope*, o qual identifica o arquivo XML como sendo uma mensagem SOAP. Um elemento *Header* que possui dados de cabeçalho. Um elemento *Body* responsável por armazenar informações de chamadas e respostas e um elemento *Fault* utilizado para manter informações e *status* de possíveis erros. A Figura 1 exemplifica a estrutura de um documento 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:Header>
...
</soap:Header>
<soap:Body>
  ...
  <soap:Fault>
    ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

Figura 1: Estrutura de um documento SOAP

3.1.2 WSDL

Quando se cria um serviço, geralmente se espera que outras pessoas o utilizem. Para que isso seja possível, o consumidor do serviço deve conhecer quais as informações devem ser enviadas ao serviço, que informações são retornadas por ele e onde encontrá-lo. Ter um formato padronizado para essas informações torna o consumo de um serviço mais simples. Foi para este fim que a WSDL *(Web Service Description Language)* foi adotada.

Conforme Weerawarana, et al. (2005), esta linguagem de descrição foi criada no ano 2000, originada da combinação de duas linguagens: NASSL (*Network Application Service Specification*) da IBM e SDL (*Service Description Language*) da Microsoft. No ano seguinte a versão 1.1 foi enviada como nota para W3C e em 2007 tornou-se recomendação em sua versão 2.0. A linguagem WSDL é considerada um vocabulário XML utilizado para descrever e localizar *Web services.* Permite que desenvolvedores de serviços disponibilizem informações importantes para a utilização dos mesmos. É altamente adaptável e extensível, o que permite a descrição de serviços que se comunicam por diferentes meios, tais como SOAP, RMI/IIOP.

Segundo Potts (2003), o documento WSDL é dividido logicamente em duas áreas, as descrições abstratas e as concretas, ilustrado pela Figura 2. A parte abstrata descreve a capacidade do *Web service*, como mensagens recebidas e enviadas pelo mesmo. Na parte concreta são descritos os elementos utilizados para vincular fisicamente o cliente ao serviço.

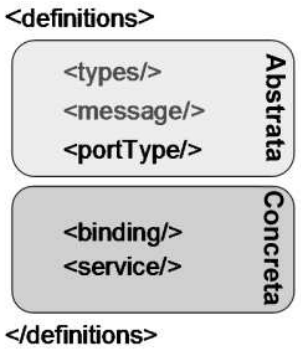


Figura 2. Divisão lógica de um documento WSDL 1.1

Na versão 2.0 da especificação WSDL, algumas mudanças foram feitas em relação à versão 1.1. Uma das principais mudanças foi a retirada do elemento `message`. Sem este elemento, a referência ao tipo de dado enviado ou recebido em uma mensagem fica no elemento `operation`, o qual referencia diretamente um tipo de dado no elemento `type`. Outras mudanças na estrutura da linguagem dizem respeito a renomeação de elementos: `definitions` para `description`; `portType` para `interface` e; `port` para `endpoint` [Weerawarana 2006].

3.1.3 UDDI

Description, Discovery, and Integration (UDDI) é um protocolo que disponibiliza métodos padrões para publicação e localização de informações sobre *Web services*, funcionando como um repositório de metadados com informações úteis para a utilização destes serviços**.** Em um cenário onde uma empresa utiliza somente seus próprios *Web services* para a execução de diferentes processos, ou compartilha serviços entre poucas companhias, o gerenciamento destes não envolve muita complexidade. Com o passar do tempo, a gama de parceiros de uma empresa pode vir a crescer e mais serviços em comum serão utilizados pelas mesmas. No processo de uma venda via comércio eletrônico, onde companhias utilizam não somente os seus serviços, mas o de outras companhias para realizar uma transação bancária ou a atualização de um estoque, a necessidade de um registro de serviços universal, padronizado, se torna fundamental para a busca e cadastro dos mesmos **[**CHAPPELL 2002**]**. Este registro é conceitualmente único, mas fisicamente dividido em diferentes nodos, os quais replicam seus dados sincronizando uns com os outros. Estes nodos ficam hospedados em empresas como Microsoft e IBM ligadas à internet.

3.2 *Web Services* REST

REST (*Representational State Transfer*) é um termo que foi utilizado pela primeira vez por Roy Fielding (um dos criadores do protocolo HTTP), em sua tese de doutorado publicada no ano 2000. Este estilo de arquitetura de software para sistemas distribuídos (como a *World Wide Web*) não se torna aplicável somente no desenvolvimento de *Web services*.

O termo é geralmente usado para descrever qualquer interface que transmita dados de um domínio específico sobre HTTP sem uma camada adicional de mensagem como SOAP ou *session tracking* via *cookies* HTTP. Estes dois conceitos podem entrar tanto

em conflito como em sobreposição. É possível desenvolver um sistema de software de acordo com as restrições impostas pelo estilo arquitetural REST sem usar HTTP e sem interagir com a Web. Também se torna possível projetar interfaces HTTP + XML que não condizem com os princípios REST de Fielding [Fielding 2000]. Sistemas que seguem os princípios REST são referenciados também como "*RESTful*".

3.2.1 Princípios do estilo REST

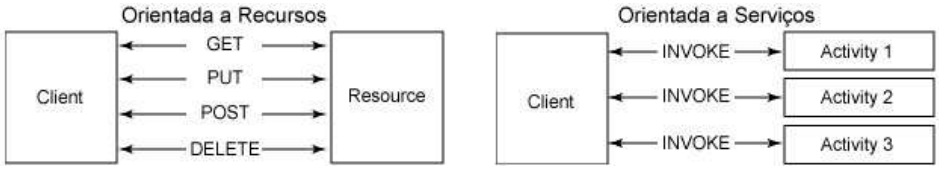
Segundo [Fielding 2000], para que os princípios deste estilo sejam respeitados, um conjunto de restrições deve ser seguido, os quais são abordados em detalhes a seguir.

• **Cliente-Servidor**: esta característica é mais comumente encontrada em aplicações Web. Um servidor, com um conjunto de serviços disponíveis, escuta requisições a estes serviços. Um cliente, que deseja que um serviço disponível no servidor seja executado, envia uma requisição para o servidor. O servidor então pode tanto rejeitar como executar o serviço solicitado, e retornar uma resposta ao cliente.

• ***Stateless* (Sem estado)**: outra restrição imposta pelo estilo REST diz respeito à interação entre cliente e servidor. A comunicação deve ser feita sem o armazenamento de qualquer tipo de estado no servidor, ou seja, cada requisição do cliente para o servidor deve conter todas as informações necessárias para que ela seja entendida. Portanto, estados de sessão, quando necessários, devem ser totalmente mantidos no cliente.

• **Cache**: uma forma de diminuir o impacto da desvantagem trazida pela redução de desempenho é a utilização de *cache*. O mesmo exige que os dados de uma resposta, vindos de uma requisição ao servidor, sejam marcados como *cacheable* ou *noncacheable* (passíveis ou não de utilização da *cache*). Se uma resposta é setada como *cacheable*, então ela será reutilizada como resposta para as futuras requisições equivalentes.

• **Interface Uniforme**: a característica central que diferencia o estilo arquitetural REST de outros estilos baseados em rede é sua ênfase em uma interface uniforme entre os componentes (cliente, servidor). Com o objetivo de obter uma interface uniforme, REST define quatro requisitos de interface: (i) identificação de recursos; (ii) manipulação de recursos através de representações; (iii) mensagens auto-descritivas e; (iv) hipermídia como mecanismo de estado da aplicação. A Figura 3 ilustra as interfaces em ambas as abordagens (serviços *RESTful* e WS-*).



Fonte: Snell, 2004.

Figura 3. Abordagem orientada a recursos (ROA) e abordagem orientada a serviços (SOA)

Atualmente, o HTTP é o protocolo chave, sendo o mais indicado para trabalhar com *Web services* REST. O mesmo provê uma interface uniforme com quatro métodos básicos para as quatro operações mais comuns. **GET** para recuperar uma representação

de um recurso, **PUT** para criar um novo recurso ou modificar um existente, **DELETE** para deletar um recurso e **POST** comumente utilizado para criação de um novo recurso.

• **Multicamada**: com o intuito de aperfeiçoar o requisito de escalabilidade da Internet, foi adicionado ao estilo REST a característica de divisão em camadas. Sistemas multicamada utilizam camadas para separar diferentes unidades de funcionalidade. A principal desvantagem deste modelo está na adição de *overhead* e latência nos dados processados, reduzindo a *performance*. Para um sistema baseado em rede que suporte *cache*, esta desvantagem pode ser amenizada.

• ***Code-On-Demand***: o último item do conjunto proposto pelo estilo REST é uma característica opcional. REST permite que clientes tenham a funcionalidade de baixar e executar diretamente código no lado cliente. Deste modo, prima pela simplificação da parte cliente e foca na extensibilidade, em contrapartida, reduz a visibilidade. Um exemplo prático conhecido de *Code-On-Demand* é o *Adobe Flash*: Um usuário (cliente) solicita uma página *Web* a qual contém um link para um *SWF* usando um *web browser*. Após a requisição, a página *Web* é transportada para a máquina do cliente juntamente com um *SWF* e o mesmo é executado.

3.2.2 Troca de Mensagens

Uma das idéias centrais na troca de mensagens é a utilização de um URI de identificação única para cada recurso. Dependendo do método utilizado para invocá-lo e dos dados na requisição HTTP, este URI terá um funcionamento diferenciado. Para melhor contextualizar pode-se tomar como exemplo um *Web service* de uma loja virtual qualquer, onde existam serviços para gerenciamento de produtos (Figura 4), com funções básicas como cadastro, alteração, exclusão e listagem de produtos. Para invocar um destes serviços REST identificados por URIs, utiliza-se o formato genérico de mensagem HTTP. Caso seja feita uma solicitação a um serviço de exclusão ou listagem de produto (DELETE, GET), apenas a requisição para o identificador do serviço é necessário, pois o método HTTP já indica qual a operação a ser realizada e o recurso solicitado. Caso seja solicitada uma requisição de cadastro de produto ou de alteração do mesmo (POST, PUT), serão enviados no corpo da requisição, os dados do produto, como: nome, descrição e marca.

```
1  <produtos>
2     <produto>
3        <categoria>
4           <descricao>Livros</descricao>
5           <idcategoria>1</idcategoria>
6        </categoria>
7        <descricao>Biografia do CEO da Apple</descricao>
8        <fabricante>
9           <descricao>Editora Nacional</descricao>
10          <idmarca>1</idmarca>
11          <nome>Agir Editora Ltda</nome>
12       </fabricante>
13       <idproduto>1</idproduto>
14       <nome>A Cabeça de Steve Jobs</nome>
15    </produto>
16 </produtos>
```

	Value
Date	Sun, 31 May 2009 22:29:15 GMT
#status#	HTTP/1.1 200 OK
Content-Length	402
Content-Type	text/xml

Figura 4. Retorno XML de um Web Service RESTful

Como protocolo do exemplo, foi definido que, para buscar os dados de um produto, seria necessário passar o seu ID como parâmetro. Não informando este ID, todos os produtos seriam listados. Um exemplo de retorno XML de uma chamada a um destes serviços pode ser visto na Figura 4.

Outra característica do protocolo HTTP, muito útil na troca de mensagens nos serviços REST, é o retorno de códigos de *status* na resposta a uma requisição. Voltando ao exemplo do cadastro de um produto, pode-se ter como retorno um "HTTP/1.1 201 *Created"* para a chamada ao serviço de cadastro caso o serviço seja executado com sucesso, ou um "HTTP/1.1 404 *Not Found"* para a busca de um produto específico que não foi encontrado. O HTTP é simples e de baixo *overhead* e sabendo utilizar bem suas características, o desenvolvimento de *Web services* REST se torna ainda mais simples [Silva 2008].

3.2.3 Representações

Quando uma aplicação é dividida em recursos, conforme pregado pelo estilo REST, as necessidades do usuário de um serviço podem ser sanadas de uma forma mais dinâmica. O usuário pode construir uma URI apropriada e acessá-la para que o serviço desejado seja alcançado. Trabalhando com a idéia de recursos torna-se possível oferecer representações em diferentes formatos e linguagens. Um usuário que acessa um serviço da Itália, por exemplo, pode receber uma representação de um recurso em italiano, como um americano, que acessa o mesmo serviço dos EUA, pode ter como resultado a representação na língua inglesa.

Um recurso é uma fonte de representações, e uma representação são apenas dados sobre o estado do recurso corrente. A maioria dos recursos na *Web* são itens de dados próprios, como uma lista de produtos, logo uma representação óbvia de um recurso são seus dados em si. O servidor pode então oferecer uma lista de produtos como um documento XML, uma página *Web* ou em formato texto separado por vírgula.

Caso o servidor ofereça múltiplas representações de um recurso, o cliente do serviço deve decidir pela representação desejada. Existem diferentes maneiras *RESTful* de passar essa informação ao servidor. Uma das formas mais simples propõe a criação de URIs distintas para cada representação de um recurso. Na analogia de um *Web service* que lista produtos de uma loja, pode-se consumir o serviço tendo como retorno uma representação em XML pela URI **http://www.loja.com.br/produtos/lista.xml** e uma representação em formato JSON pela URI **http://www.loja.com.br/produtos/lista.json.**

Uma segunda alternativa é chamada de *content negociation.* Neste cenário, existiria somente uma URI para representar a lista de produtos, **http://www.loja.com.br/produtos/lista,** no momento que um cliente fizesse uma requisição ao serviço, juntamente a requisição, seriam enviados dados no cabeçalho HTTP, o qual sinalizaria o tipo de representação que o cliente espera receber. As informações no cabeçalho HTTP podem tanto originar do browser do cliente como também serem setadas na criação do *request*, pelo cliente consumidor do serviço. O campo *Accept-Language* do cabeçalho HTTP pode ser usado pelo servidor para identificar em que formato o cliente espera o retorno da requisição ou o campo *Content-Type* para fins de identificação do formato solicitado.

3.2.4 Descritores de *Web services* REST

Existem opiniões variadas sobre as interfaces de descrição de serviços REST. Há quem julgue que elas não são necessárias. Outros acham válido possuir um descritor de serviços, semelhante ao WSDL dos serviços WS-*. Atualmente, os *frameworks* e ferramentas voltadas ao desenvolvimento de serviços REST estão se preocupando em incluir o suporte a uma linguagem de descrição. Umas das linguagens mais difundidas disponíveis para isso é a WADL. Com a liberação da WSDL 2.0 tornou-se possível também utilizá-la para descrever um serviço REST.

4 Estudo de Caso

Nesta seção, é apresentado um estudo de caso do desenvolvimento de serviços de uma loja virtual fictícia, a qual possui *Web services* para a manipulação de informações básicas de uma loja, como: produtos, categorias e marcas. Estes *Web services* realizam manipulações de informações como inserção, edição, exclusão e consulta. Os mesmos serviços foram implementados em duas arquiteturas diferentes, uma utilizando WS-* e outra no estilo REST (ROA).

4.1 Ambiente e Configurações

Ambos os projetos foram desenvolvidos utilizando Java 6 na IDE Eclipse Ganymede 3.4.1, em que acessam uma base de dados em comum que utiliza como SGBD a versão 5 do MySQL. A estrutura de dados da base utilizada é apresentada na Figura 5. A camada de persistência dos *Web services* foi desenvolvida utilizando o *framework* Hibernate versão 3, o qual possibilita o mapeamento das entidades da base de dados em objetos Java. O servidor Web Java utilizado foi o Tomcat versão 6.

Figura 5. Estrutura de tabelas do estudo de caso

Para a criação dos *Web services* WS-* foi utilizada a API Java JAX-WS 2.0 (*Java API for XML Web Services*) a qual é baseada em *annotations* introduzidos na Java SE 5. Esta API faz parte da plataforma Java EE da Sun Microsystems e facilita o desenvolvimento e organização dos serviços.

Na criação dos *Web services* REST, foi empregada a API JAX-RS (*Java API RESTful Web Services*) que dispõe de suporte na criação de serviços de acordo com os princípios do estilo de arquitetura REST. Esta API também faz uso de *annotations* Java. Após o *deploy* das aplicações no servidor Tomcat são disponibilizados arquivos de descrição dos serviços oferecidos (WSDL, WADL), conforme Figura 6.

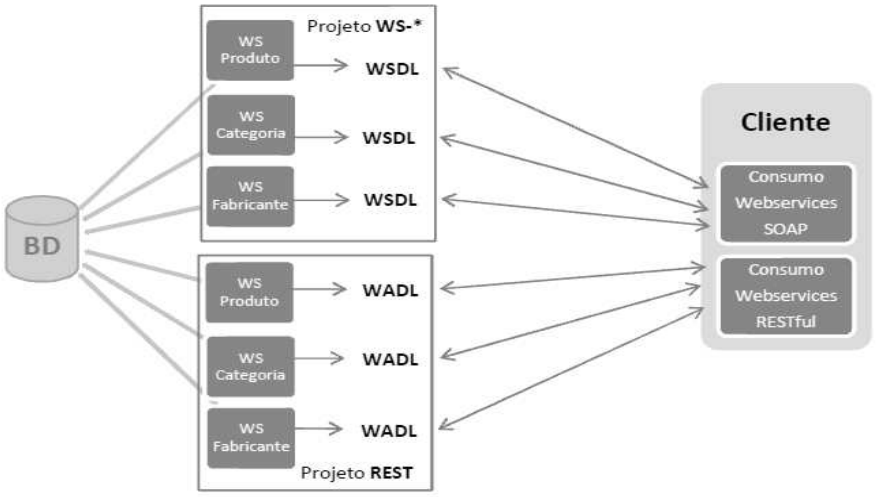


Figura 6. Fluxograma do estudo de caso

Para a criação do lado cliente, foi utilizada a ferramenta SoapUI versão 3. Esta ferramenta possibilita a criação dos consumidores dos serviços através de seus arquivos de descrição (WSDL/WADL). O preenchimento de um *header* HTTP ou o preenchimento de um envelope SOAP também é facilmente realizado.

4.2 Funcionamento

O fluxo de funcionamento dos serviços WS-* inicia com a requisição de um cliente. Esta requisição é encapsulada em um envelope SOAP que por sua vez é transportada até o *Web service* desejado em um pacote HTTP. Todos os dados que o serviço espera receber de um cliente estão descritos no arquivo WSDL, disponibilizado pelo *Web service.* A requisição de um cliente chega ao serviço através de um *Servlet* Java que serve como *Listener* de requisições e pertence ao JAX-WS.

O *Web service,* após receber a requisição que contém o envelope SOAP, o processa, executa a regra de negócio contida no corpo do serviço e monta um envelope SOAP de resposta. Esta resposta é enviada através do *dispatcher* presente na estrutura do *framework* JAX-WS para o cliente consumidor do serviço. Por fim, o cliente desmembra o SOAP retornado e analisa os dados de retorno. Eventuais falhas na execução do serviço são relatadas ao consumidor do serviço através do envelope SOAP, campo *fault*.

Nos serviços REST, o fluxo de funcionamento se diferencia em alguns pontos importantes. O cliente consumidor do *Web service RESTful* estrutura sua requisição em uma mensagem HTTP. Esta mensagem HTTP pode conter um corpo de dados que são enviados juntamente aos dados do cabeçalho. No cabeçalho pode ser informado através do campo *Content-Type,* o tipo de dado que se deseja receber do serviço (XML, JSON), caso o mesmo seja um serviço de listagem de dados. Para indicar que serviço se deseja chamar em um *Web service,* utiliza-se as URIs dos serviços invocadas utilizando um dos métodos HTTP.

A requisição do cliente chega ao *Web service* através de um *Servlet* Java (JAX-RS) que age como um *Listener* configurado no projeto REST. A partir daí, a requisição é processada, endereçada ao método solicitado, onde executa a regra de negócio contida

nesse método. Após o término da execução do método, é enviada a resposta ao cliente consumidor do serviço. Caso alguma falha tenha ocorrido na execução do método, são utilizados os códigos de erro do protocolo HTTP para informar o cliente do acontecido.

5 Análise Comparativa

No processo de análise, foi visto que algumas características estão presentes em ambas as partes. Dentre elas está a possibilidade de uso de *cache* e a propriedade de dividir o sistema em camadas, fortalecendo a reusabilidade de código. A Tabela 1 apresenta itens da comparação entre serviços WS-* e serviços *RESTful*.

Tabela 1. Análise Comparativa entre Web Services WS-* e Web services REST

Fator de Comparação		***Web services* WS-***	***Web services* REST**
Tamanho de Pacote HTTP	WS lista produtos por categoria	584 bytes	399 bytes
	WS lista produtos	1567 bytes	1412 bytes
	WS lista fabricantes	572 bytes	431 bytes
	WS lista categorias	864 bytes	747 bytes
Armazenamento de estados		Permite	*Stateless*
Abordagem de design		Baseado em operações ou verbos	Baseado em recursos ou substantivos (URIs)
Interface		Interface não uniforme	Interface uniforme – GET, POST, UPDATE, DELETE.
Code on demand (COD)		Não é voltado a COD	Permite que código originado da requisição ao Servidor seja executado no lado Cliente.
Representação de dados		XML	Aberto
Descritor		WSDL	WSDL/WADL/Nenhum

5.1 Fatores de comparação

Os fatores de comparação utilizados para a análise do estudo de caso foram:

- **Tamanho do pacote HTTP**: indica a dimensão das mensagens trocadas entre um *Web service* e seu consumidor, ou seja, esboça uma amostra do tráfego gerado na conversa entre os dois pontos. O protocolo HTTP foi escolhido como um dos itens de comparação, pois é bastante difundido, utilizado como protocolo de transporte em serviços WS-* e base no funcionamento dos serviços REST.
- **Armazenamento de estados:** reflete como as abordagens se comportam em relação à possibilidade de guardar estados. Este fator, quando presente, possibilita um ganho de desempenho, pois mantêm estados para serem utilizados em requisições posteriores.

- **Abordagem de design:** indica de que forma são vistos os serviços, de que maneira estão dispostos os serviços para o consumidor.
- **Interface**: especifica a forma de iteração com os serviços, a maneira a qual é requisitada uma determinada operação oferecida por um serviço.
- ***Code on demand***: quando presente, indica a possibilidade de se estender características ou funcionalidades de um cliente, recebendo códigos que serão executados no mesmo.
- **Representação de dados**: trata dos formatos que um serviço pode prover ao requisitante. Possibilita fazer a escolha de uma das abordagens com base no ambiente em que os serviços estarão engajados.
- **Descritor**: especifica as possíveis linguagens de descrição que um serviço pode oferecer. Este fator comparativo é de suma importância, pois através de um arquivo de descrição torna-se possível automatizar etapas no desenvolvimento e consumo de serviços.

5.2 Análise

Os pacotes utilizados na comparação foram pacotes de resposta a serviços de listagem de produtos, produtos por categorias, fabricantes de produtos e categorias de produtos. Foi utilizado o formato XML como retorno das chamadas em ambas as abordagens de serviços, sendo seus tamanhos calculados pela ferramenta SoapUI 3. Nesse aspecto, REST leva vantagem, uma vez que, se utiliza somente do HTTP para transportar as informações, diferente dos serviços WS-* que trabalham com envelopes SOAP envolvidos por pacotes HTTP. Com isso o desempenho também é afetado, pois os pacotes SOAP devem sofrer um *parse* antes de os dados serem utilizados. A Figura 7 apresenta um gráfico com os resultados obtidos na comparação.

As diferenças variaram de 117 a 185 bytes em virtude da quantidade de dados identificados no retorno dos *Web services*, porém, sempre vai haver acréscimos em virtude do protocolo SOAP.

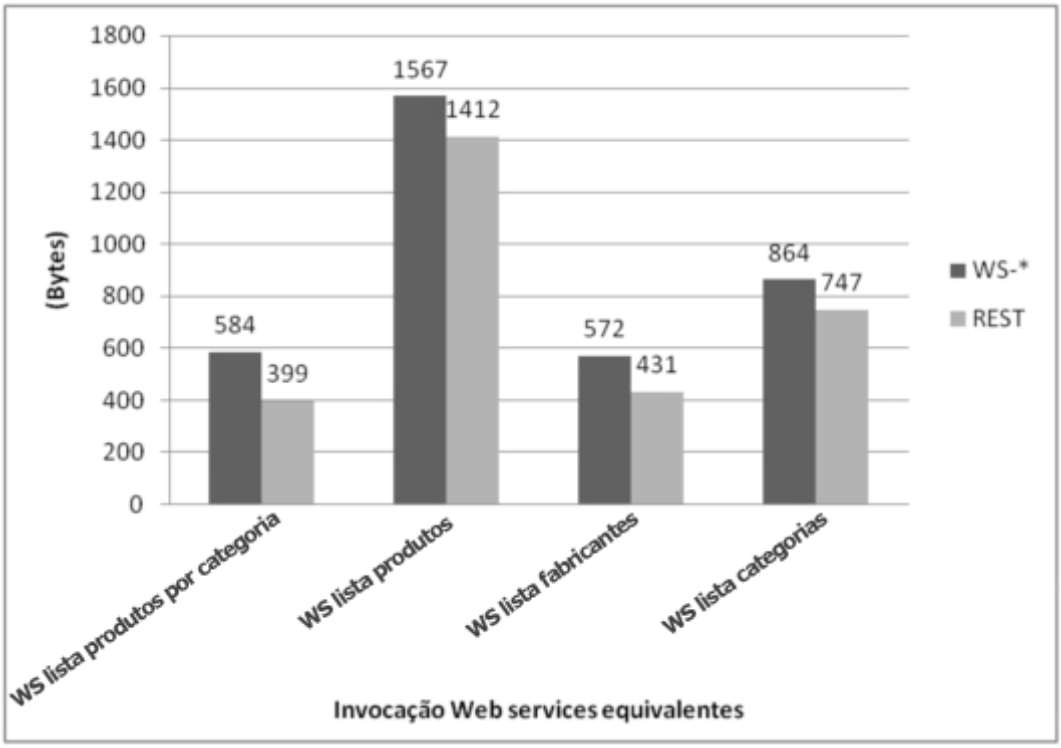


Figura 7. Tamanho de pacote HTTP em *Web services* WS-* e REST

Na comunicação Cliente/Servidor, nenhum tipo de estado é mantido (*stateless*) no uso de *RESTful*. Esta característica prima pela escalabilidade e simplicidade dos serviços e da infra-estrutura envolvida. Nos testes tanto os *Web services RESTful* quanto os WS-* foram desenvolvidos sem armazenamento de estados, mas os serviços WS-* podem ser desenvolvidos visando esta característica. Por exemplo, em uma aplicação que necessite de autenticação, a qual solicite um usuário e uma senha, o cliente do serviço *RESTful* deve enviar os dados de autenticação a cada requisição. Nos serviços WS-* é possível solicitar os dados de autenticação somente na primeira requisição, salvando os mesmos em sessão no servidor. Nesta abordagem, pode-se ganhar em desempenho, pois estes dados não precisarão ser enviados e validados a cada requisição. Por outro lado, a questão da escalabilidade se torna mais complexa e custosa onde o armazenamento de estado de múltiplos clientes acarretará o aumento na demanda de recursos como memória e processamento no servidor.

Enquanto a abordagem dos serviços WS-* disponibiliza operações tais como *getCategoria()*, *setCategoria()*, *deleteCategoria()*, os serviços *RESTful* se baseiam em recursos representados por URIs. Para requisitar os métodos REST equivalentes citados acima é necessário requisitar a URI “http://localhost:8888/categorias/[nomeCategoria]” requisitada juntamente com os verbos da interface uniforme do HTTP, ou seja, uma requisição HTTP/GET para esta URI busca os dados dessa categoria, uma requisição POST cria uma nova categoria e uma requisição DELETE a deleta.

Os verbos do HTTP caracterizam a interface uniforme presente no estilo REST. Sendo que nos serviços WS-* a interface não é uniforme e é definida pelo desenvolvedor quando nomeia as atividades do serviço (nome dos métodos). Com uma interface uniforme o desenvolvimento fica mais claro e padronizado, entretanto, as

URIs podem não ter o mesmo poder de descrição de um nome de método bem elaborado. Outro ponto negativo em possuir uma interface uniforme está na diminuição da eficiência, pois a transmissão dos dados se dá de uma forma padronizada ao invés de ser otimizada para as necessidades de uma aplicação específica.

A possibilidade de baixar *scripts/Applets* para execução no cliente não tem nenhuma restrição de ser feita através de serviços *RESTful* ou WS-*. É possível baixar código *Javascript* ou um *Applet* fazendo requisições SOAP, embora isso seja bastante incomum, sendo mais factível ocorrer com REST do que com WS-*. REST é freqüentemente associado a Web, onde o navegador é fortemente utilizado, provendo um melhor suporte a *Code on Demand*, comparado aos serviços SOAP que são mais associados a *Desktop*. Os navegadores possibilitam que um servidor retorne código para ser executado no lado cliente. Essa execução adicional de código permite estender ou até mesmo customizar a capacidade de um cliente sem a necessidade de instalações ou alterações. Sendo um exemplo prático o objeto XmlHttpRequest do Javascript (AJAX). Com base no *Code on Demand* desenvolvido no estudo de caso dos serviços REST pode-se notar que apesar de trazer vantagens como as citadas acima, esta prática encobre o que está sendo recebido e executado no cliente, logo, deve-se confiar na fonte do serviço.

A possibilidade de retornar dados em diferentes representações torna os serviços mais dinâmicos e oferece diferentes opções ao cliente requisitante. Esta característica dos serviços REST não é encontrada nativamente nos serviços WS-*, os quais trabalham com representação XML por padrão. Nos serviços desenvolvidos no estudo de caso, o retorno dos *Web services* no estilo REST podem ser requisitados nas representações XML ou JSON, sendo que em serviços WS-*, representações diferentes da XML não são possíveis sem serem parte do conteúdo do pacote SOAP.

A utilização de um arquivo descritor de *Web service* é de fundamental importância para serviços WS-*. Através dele, ferramentas de desenvolvimento e até mesmo os próprios programadores têm acesso a informações úteis sobre os *Web services*. Estes descritores podem auxiliar no consumo dos serviços, tornando este mais viável. Os *Web services RESTful* podem ser descritos através de WSDLs ou WADLs, mas seu uso não é um requisito imposto pelo estilo REST. Esta flexibilidade garante que um serviço *RESTful* funcione normalmente sem um arquivo descritor. No estudo de caso descrito no artigo, foi utilizada WSDL para descrever os serviços WS-* e WADL para os *RESTful*, com isso o consumo dos serviços através da ferramenta SoapUI se tornou prática e fácil. Sem a utilização de um descritor o consumo de um serviço ou criação de um cliente para este serviço se tornaria mais demorado, tendo que o desenvolvedor buscar manualmente as informações necessárias. Ferramentas como a IDE JDeveloper da Oracle, NetBeans da Sun Microsystem e Eclipse inicialmente desenvolvido pela IBM, utilizam dos descritores para a geração de serviços ou clientes de forma automatizada.

6 Conclusões

Este artigo apresentou uma análise comparativa prática e conceitual entre *Web services RESTful* e WS-*. Também apresentou um estudo de caso para demonstrar as duas abordagens e servir de instrumento no processo de comparação. No decorrer dos estudos, constatou-se que as duas abordagens têm suas vantagens e desvantagens,

demonstrando que cada uma tem sua área de especialidade e, portanto, o dever de coexistir.

Os serviços WS-* têm a vantagem de serem bastante difundidos atualmente nas empresas e organizações de grande porte, ideais em circunstâncias que envolvam diferentes protocolos de comunicação; em ferramentas *middleware* utilizadas na integração de sistemas complexos e de diferentes arquiteturas (BPEL, ESB). Serviços *RESTful*, por outro lado, têm uma boa resposta em aplicações que envolvam manipulações de dados onde resgatam as características da *Web*. Recomenda-se em aplicações *mobile*, onde o custo na troca de mensagens é bastante elevado; em blogs e sites de relacionamento, os quais são baseados em recursos e demandam de chamadas RSS ou *Atom*; em serviços de busca na internet, que trabalham com um alto tráfego de informações.

Como trabalhos futuros, têm-se em vista a avaliação de ferramentas para o desenvolvimento de *Web services* que suportem ambos os tipos de serviços, bem como o detalhamento e aprofundamento de metodologias para implementar segurança em serviços de ambas as abordagens.

Referências

CHINTAKHA, Eran. Enable REST with Web services, Part 1: REST and Web services in WSDL 2.0. maio 2007. Disponível em: < http://www.ibm.com/developerworks/webservices/library/ws-rest1/>. Acesso em: 09 jan. 2010.

FIELDING, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. 2000. Tese (Doutorado em Informação e Ciência da Computação) – Universidade da Califórnia, Califórnia, 2000.

MENÉNDEZ, Andrés Ignácio Martínez. Uma ferramenta de apoio ao desenvolvimento de Web Services. Dissertação de Mestrado, Universidade Federal de Campina Grande, curso de Pós-Graduação em Informática, 2002.

MUEHLEN, Michael zur; NICKERSON, Jeffrey V.; SWENSON, Keith D. Developing web services choreography standards—the case of REST vs. SOAP. Decision Support Systems. V.40, n.1, July 2005, Pages 9-29

NEWMARCH, Jan; A RESTful Approach: Clean UPnP without SOAP: Using Java in Service-Oriented Architectures. *Consumer Communications and Networking Conference*, 2005. CCNC. 2005 Second IEEE: 134-138, Jan. 2005.

PAUTASSO, Cesare. SOAP vs. REST: Bringing the Web back into Web Services. IBM Zurique Research Lab. 2007. Technical Report. Disponível em: http://www.iks.inf.ethz.ch/education/ss07/ws_soa/slides/SOAPvsREST_ETH.pdf. Acesso em: jan/2010

PAUTASSO, Cesare. REST vs WS-* Comparison. Tutorial at WWW 2009, PART IV. Disponível em: < http://www2009.eprints.org/253/1/rest-ws.pdf>. Acesso em: 18 set. 2009.

PRESCOD, Paul. Roots of the REST/SOAP Debate, Proceedings of Extreme Markup Languages Conference. Montréal, Canada, August 2002.

SHI, Xuan; Sharing service semantics using SOAP-based and REST Web services. *IT Professional* (8): 18-24, mar./abr. 2006.

SILVA, Pereira Bruno Luiz. REST vs WS-*: Uma Visão Pragmática. Java Magazine, São Paulo, Edição 54, pág. 38 à 47, 2008.

SNELL, James. Resource-oriented vs. activity-oriented Web services. 2004. Disponível em: <http://www.ibm.com/developerworks/webservices/library/ws-restvsoap/>. Acesso em: 15 set. 2009.

STREIBEL, Martin. Implementando Web Services. dez. 2005. Disponível em: <http://palazzo.pro.br/artigos/martin.htm>. Acesso em: 10 mai. 2009.

WEERAWARANA, Sanjiva; CURBERA, Francisco; LEYMANN, Frank; STOREY, Tony; FERGUSON, Donald F. Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Adressing, WS-BPEL, WS-Reliable Messaging, and more. Prentice Hall PTR, 2005.

W3C. Simple Object Access Protocol (SOAP) 1.1: W3C Note 08 May 2000. Disponível em: < http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>. Acesso em: 14 jun. 2009.

W3C. Web Services Architecture: W3C Working Group 2004. Disponível em: <http://www.w3.org/TR/ws-arch/ >. Acesso em: 13 maio 2009.