UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

***CAMPUS* LUIZ MENEGHEL**

AUGUSTO TOBIAS NETO

COMPARAÇÃO VIA SOFTWARE ENTRE DOIS MODELOS DE WEB-SERVICES VISANDO O DESEMPENHO E LIMITAÇÃO DAS APLICAÇÕES

Bandeirantes

2012

AUGUSTO TOBIAS NETO

COMPARAÇÃO VIA SOFTWARE ENTRE DOIS MODELOS DE WEB-SERVICES VISANDO O DESEMPENHO E LIMITAÇÃO DAS APLICAÇÕES

Trabalho de Conclusão de Curso apresentado à Universidade Estadual do Norte do Paraná – *campus* Luiz Meneghel – como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Estevan Braz Bandt Costa

Bandeirantes

2012

AUGUSTO TOBIAS NETO

COMPARAÇÃO VIA SOFTWARE ENTRE DOIS MODELOS DE WEB-SERVICES VISANDO O DESEMPENHO E LIMITAÇÃO DAS APLICAÇÕES

Trabalho de Conclusão de Curso apresentado à Universidade Estadual do Norte do Paraná – *campus* Luiz Meneghel – como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

COMISSÃO EXAMINADORA

_____
Prof. Estevan Braz Bandt Costa Orientador
UENP – *Campus* Luiz Meneghel

_____
Prof. Me. Ricardo Gonçalves Coelho
UENP – *Campus* Luiz Meneghel

_____
Prof. Luiz Fernando Legore do Nascimento
UENP – *Campus* Luiz Meneghel

Bandeirantes, __ de_____ de 2012

AGRADECIMENTOS

Primeiramente à Deus, pois sem ele eu não estaria aqui, e não teria conseguido atingir meus objetivos. A toda minha família, que sempre me apoiou e lutou em todos os momentos ao meu lado, para que conseguisse concretizar meus estudos, há quem muito devo por ter chegado aqui, fica meu eterno sentimento de amor e gratidão.

À minha esposa Alvilene e minha filha Sarah, que tanto sofreram com minha ausência quando da elaboração deste TCC e dos diversos trabalhos durante os quatro anos do curso. Mais ciente que tudo isso é para eu poder lhes proporcionar um futuro melhor.

Aos meus pais Marcos e Rosana, que sempre me ensinaram a ser uma pessoa honesta, digna e de caráter, um exemplo em minha vida.

Aos meus avós Antônio e Iraci, que ajudaram muito, incentivando e contribuindo para que conseguisse realizar o sonho de se formar.

Aos meus irmãos Marienne e Marcos, que sempre com muito amor e carinho estiveram ao meu lado para dar forças nesta minha formação.

Aos meus amigos e colegas, que estiveram presentes nessa caminhada, sempre ao meu lado, enfrentando junto a mim as dificuldades de um curso superior, fica o meu obrigado pelo companheirismo e amizade.

Ao meu orientador e demais professores da UENP – Campus Luiz Meneghel que me ensinaram, além de todas as disciplinas, a ser um profissional ético e competente.

Às demais pessoas que contribuíram direta ou indiretamente para o meu sucesso.

Para quem só tem um
martelo, qualquer coisa
vira prego.
(Roberto Banaco)

RESUMO

Web Service pode ser definido como sendo uma aplicação web e tem por finalidade facilitar a comunicação entre aplicações de plataformas distintas. O que vem motivando a utilização de *Web Services* são as inúmeras possibilidades oferecidas por esta tecnologia, ainda mais quando consideramos a abrangência e o alcance da web. Existe uma opção para tal problema que é a utilização de um *software* que realiza teste de carga que irá testar estes *Web Services* e colher os resultados, desta forma será possível afirmar se SOAP ou REST é mais eficiente. A necessidade deste projeto leva a sanar as dúvidas na hora de escolher qual é a implementação mais viável de se utilizar, levando-se em conta a facilidade e limitação da aplicação. O presente trabalho tem por objetivo desenvolver dois *Web Services* utilizando diferentes formas de implementação, ambos desenvolvidos por tecnologias livres.

Palavras-chave: *Web Service*; Arquitetura; SOAP; REST.

ABSTRACT

Web Service can be defined as a web application and is intended to facilitate communication between applications of different platforms. What is motivating the use of Web Services is the numerous possibilities offered by this technology, especially when we consider the scope and reach of the web. There is an option for this problem is to use software that performs load test that will test these Web Services and harvest results thus be possible to say whether SOAP or REST is more efficient. The need for this project takes to remedy the doubts in choosing which implementation is more viable to use, taking into account the limitations and ease of application. This work aims to develop two Web Services using different forms of implementation, both developed by open technologies.

Keywords: *Web Service; Architecture;* SOAP; REST.

LISTA DE FIGURAS

LISTA DE QUADROS/GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

CRUD	*Create, Read, Update e Delete*
HTML	Linguagem de Marcação de Hipertexto
HTTP	Protocolo de Transferência de Hipertexto
IDE	Ambiente Integrado de Desenvolvimento
JSON	Notação de Objetos JavaScript
JVM	Máquina Virtual Java
MIME	Extensões Multi função para Mensagens de Internet
MYSQL	Linguagem de Consulta Estruturada
PDA	Assistente Pessoal Digital
QoS	Qualidade de Serviço
REST	Transferência de Estado Representativo
SOAP	Protocolo Simples de Acesso ao Objeto
TCC	Trabalho de Conclusão de Curso
UML	Linguagem de Modelagem Unificada
URI	Identificador Uniforme de Recursos
W3C	*World Wide Web Consortium*
WADL	Linguagem de Descrição de Aplicações Web
WSDL	Linguagem de Descrição de Serviços Web
XML	Linguagem de Remarcação Extensível

SUMÁRIO

1 INTRODUÇÃO

A Internet está cada dia crescendo mais, e sendo utilizada para diversos fins. Logo, este meio se tornou o foco do desenvolvimento de novos aplicativos, que há pouco tempo somente era utilizado para buscas de conhecimento e informação. Nesses aplicativos os quais deve-se criar novos métodos de desenvolvimento, que possa fazer com que os serviços prestados sejam interoperáveis e também reutilizáveis.

Através desta necessidade o objetivo das empresas é de adaptar seus *software* já desenvolvidos para que estes sofram alterações e que, após ser reescrito possam ser aplicados neste cenário atual.

Uma tecnologia que faz este serviço é o *Web Service*, o qual tem como objetivo integrar sistemas. Pois como asseguram Hansen e Pinto (2003, p. 1) “os *Web Services* apresentam uma estrutura arquitetural que permite a comunicação entre aplicações”. Dessa forma os *Web Services* conseguem disponibilizar serviços e funções aos seus clientes, como também incorporar e reutilizar outros serviços disponíveis em outros servidores.

A comunicação é feita por meio de troca de mensagens XML (*eXtensible Markup Language*), pois esta linguagem possui o poder de armazenar todo o tipo de dado e ainda descrevê-los. Dessa forma ele passa a ser totalmente transparente para o cliente que esta solicitando determinada operação.

Atualmente existem vários tipos de *Web Services*, dentre eles pode-se citar os dois mais populares, SOAP e REST.

Uma arquitetura de *Web Service* que está ganhando destaque por sua simplicidade e confiabilidade de implementação e utilização é o REST (*Representational State Transfer*), o qual faz uso dos recursos oferecidos pelo protocolo HTTP (*Hypertext Transfer Protocol*).

Além disso, o REST oferece mais agilidade e facilidade aos usuários e desenvolvedores permitindo a construção de aplicações mais leves e com alto desempenho segundo Santos (2009).

O presente trabalho visa apresentar alguns exemplos de implementação em Java desse tipo de *Web Service*, chamada RESTEasy. Essa implementação é feita

utilizando a especificação JSR-311 ou JAX-RS. Além disso, propõe-se demonstrar a simplicidade de manipulação do mesmo, justificando o fato deste ser tão popular, e como ele está ganhando força no mercado cada vez mais.

1.1 CONTEXTO E DELIMITAÇÃO DO TRABALHO

O desenvolvimento de *Web Services* pode ser bem complexo, e a utilização de alguns fatores pode ajudar nesta tarefa. O presente trabalho irá produzir e realizar testes de *Web Services* utilizando bibliotecas diferentes.

1.2 FORMULAÇÃO DO PROBLEMA

Será que compensa desenvolver *Web Services* utilizando a biblioteca RESTEasy?

1.3 HIPÓTESE

A solução encontrada para o problema acima, é de que a tecnologia além de ajudar no desenvolvimento, ela consegue atender as necessidades primárias de um *Web Service* normal.

1.4 Objetivos

1.4.1 Objetivo Geral

Este trabalho possui como objetivo geral, analisar o processo de desenvolvimento dos *Web Services* em arquiteturas diferentes, os resultados dos testes, bem como aspectos fundamentais de tecnologias base para *Web Services*, tais como: a linguagem XML, protocolo SOAP, o descritor de *Web Services* WSDL e o estilo arquitetural REST e sua implementação RESTEasy.

Para tal objetivo será utilizado padrões da W3C, consórcio internacional cujo princípio é desenvolver padrões para a criação e interpretação de conteúdos para a web.

1.4.2 Objetivos Específicos

Os objetivos específicos do trabalho são:

a) Criar dois *Web Services*, um será desenvolvido utilizando uma arquitetura já tradicional, que é o SOAP. O segundo *Web Service* será desenvolvido no estilo arquitetural REST, que é implementado usando o RESTEasy.
b) Fornecer serviço de busca utilizando uma base de dados MYSQL;
c) Fornecer uma interface para a utilização dos serviços disponíveis;
d) Realizar testes de carga em ambos *Web Services* utilizando a ferramenta JMeter.

1.5 JUSTIFICATIVA

O tema do trabalho em questão foi escolhido pela grande dúvida na hora de escolher qual implementação é a melhor para se desenvolver um *Web Service*. Pois o uso de serviços web baseado na arquitetura REST tem crescido bastante nos últimos anos, ganhando assim evidência no meio antes dominado por arquiteturas tradicionais como é o caso do SOAP.

O estudo feito neste trabalho em cima das arquiteturas dos *Web Services* será válido e útil em dois pontos, que são eles:

1) O *Web Service* em REST, sendo implementado em RESTEasy é mais fácil de ser desenvolvido, mais simples, e José (2009, p. 8) ainda o descreve como sendo uma aplicação leve pois não tem um acúmulo de marcações extras de XML.

   Mais até que ponto esta facilidade é vantagem se comparado sua eficiência em um teste de carga? Quantas requisições ele vai aguentar sem apresentar erros, qual o tempo de resposta destas requisições. Pois dependendo do resultado não é vantagem ser implementado.

2) Agora se o resultado for interessante, quais as facilidades encontradas na sua implementação que difere com o desenvolvimento do *Web Service* construído utilizando a arquitetura SOAP. Isso poderá ser analisado se comparado à quantidade de códigos, classes, bibliotecas utilizados em ambos os casos.

Através do teste de carga será possível sanar está dúvida, e ver qual arquitetura é mais eficiente para o problema proposto. Pois o SOAP já é um padrão que está maduro no mercado, e a implementação RESTEasy está na moda, e ganhando cada vez mais evidência nos últimos anos.

O restante do trabalho está organizado da seguinte forma. O Capítulo 2 trata da fundamentação teórica conceituando *Web Service* e suas particularidades. O Capítulo 3 descreve o desenvolvimento. O Capítulo 4 traz os resultados obtidos após a realização dos testes de carga, e por fim, no Capítulo 5 é apresentado a conclusão do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção tem por objetivo apresentar os assuntos relacionados ao trabalho.

2.1 *Web Service* SOAP (*Simple Object Access Protocol*)

A Internet cada vez mais está sendo utilizada nas realizações de comunicações entre as aplicações distintas. A interface desenvolvida capaz de fazer esta comunicação chama-se *Web Services*. Ele é portanto uma tecnologia para comunicação em rede que utiliza o suporte básico da web, o protocolo HTTP, para transferência de informações.

Serviços web que funcionará como um servidor disponibilizando um ou mais serviços para seus clientes, com baixo acoplamento (SAMPAIO, 2006). Normalmente o protocolo utilizado para fazer a troca de mensagens é o SOAP.

Web Service pode ser descrito como sendo qualquer arquitetura que irá utilizar no seu transporte documentos XML (ABÍLIO E DUEIRE, 2006).

Os padrões SOAP e XML são a base para a construção de um *Web Service*. O transporte é construído totalmente via HTTP, utilizando o formato XML e sendo encapsulado via protocolo SOAP.

O *Web Service* se baseia na integração de três entidades, que são ilustradas na Figura 1, que são: provedor de serviços (*service provider*), provedor de registro (*service broker*) e cliente do serviço (*service requester*). A iteração destas entidades serve para que haja publicação, busca e execução das operações:

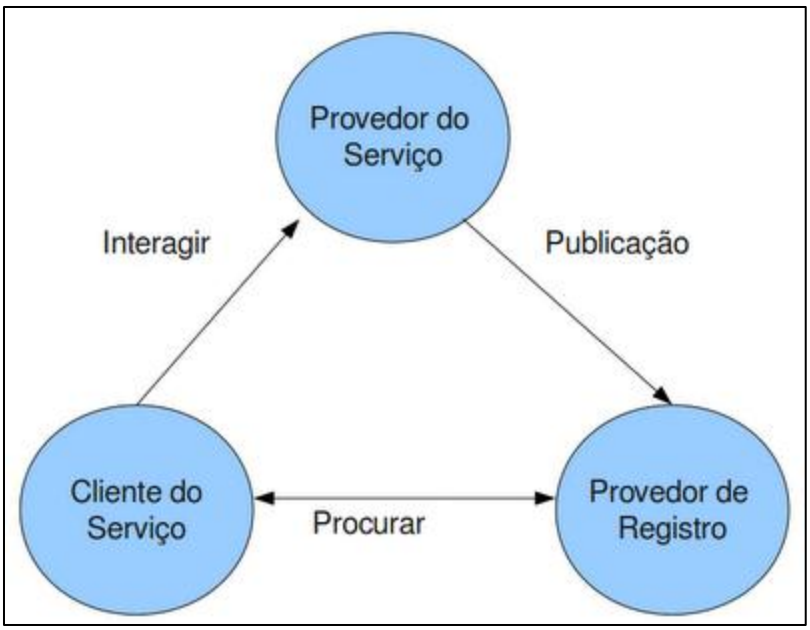

Figura 1 – Arquitetura dos *Web Services*
Fonte: Autoria própria

Para um usuário saber como utilizar os recursos de um determinado *Web Service*, é preciso usar o WSDL (*Web Service Definition Language*), linguagem criada para definir e descrever um *Web Service*. O WSDL define também operações e interfaces de um serviço.

Desta forma, tem-se os três elementos básicos que compõem os *Web Services*:

a) XML: formato para a troca de mensagens;
b) SOAP: protocolo para transporte de dados baseado em XML;
c) WSDL: descrição dos serviços e funções de um *Web Service* baseado em XML.

2.1.1 Arquitetura

Web Services são compostos por três características de concepção básica, que são definidas da seguinte maneira segundo Abílio e Dueire (2006):

- para a troca de mensagens é empregado à tecnologia SOAP;
- os metadados irão descrever os serviços disponibilizados;
- por fim, possuem diretório integrado, que irá fazer a descrição e localização de *Web Services*.

Os autores ainda afirmam que “essas três características são ordenadas

através de uma arquitetura própria orientada à serviços".

2.1.2 *eXtensible Markup Language*

Faria (2005, p. 11) descreve XML como sendo uma linguagem de marcação que tem o poder de armazenar todo tipo de dados e ainda descrevendo-os. Além de identificar os tipos de dados que possui, o padrão XML permite organizar esses dados de forma mais precisa. Esse padrão é uma recomendação da W3C, para gerar linguagens de marcação para necessidades especiais.

2.1.3 *Simple Object Access Protocol*

O SOAP foi inicialmente criado para possibilitar a invocação remota de métodos através da Internet, mais "evoluiu para ser um protocolo completo de troca de mensagens XML" (SAMPAIO, 2007).

Conforme Abílio e Dueire (2006), o SOAP tem por si só a finalidade de tornar-se uma língua franca para os *Web Services*. Este protocolo será usado por toda e qualquer comunicação e utilização de *Web Services*, pois foi projetado para ser independente de plataforma e implementado em qualquer linguagem (DEITEL, 2001).

Toda mensagem SOAP é formada por um elemento chamado envelope, e que contem os subelementos, *Header e Body*. O primeiro irá conter informações de controle. Já no *Body* irá ter o corpo da mensagem em si.

Deitel (2001) define SOAP como sendo um padrão de suma importância, dentro do setor de infraestrutura de computação distribuída XML.

2.1.4 *Web Service Definition Language*

WSDL é um padrão baseado em XML. Com esta linguagem é possível ver todos os serviços e as interfaces oferecidas por uma determinada aplicação, e também como sua localização (RECKZIEGEL, 2006).

Por ser um documento XML, sua leitura se torna fácil e acessível, através dos componentes que formam sua especificação.

Para Reckziegel (2006), os componentes deste documento são:

- Tipos (*types*): Definição de dados usados nas mensagens;
- Mensagem (*message*): define os parâmetros de entrada e saída de um serviço;
- Operações (*operation*): é a assinatura do método que faz o relacionamento entre os parâmetros de entrada e saída;
- Tipo de porta (*portype*): descreve a maneira que esta sendo feito o agrupamento lógico das operações;
- Vínculo (*binding*): uma especificação concreta de protocolo e formato de dados para um *PortType*;
- Serviço (*service*): além de todos os componentes descritos acima também irá definir um serviço.

Para uma maior compreensão destes componentes que formam um documento XML, são ilustrados no Quadro 1 seus principais elementos.

```
<definitions>
  <types>
  [ XML Schema a describind the used datatypes ]
  </types>

  <message>
  [ Message description ]
  </message>

  <portType>
    <operation>
     [ References to input and output messages ]
    </operation>
  </portType>

  <binding>
  [ Description of network protocol for
  invocation ]
  </binding>

  <service>
    <port>
     [ Reference to acual location of service ]
    </port>
  </service>
</definitions>
```

Quadro 1 – Principais elementos de um WSDL
Fonte: W3C

O WSDL como qualquer XML, também tem o seu elemento raiz, onde contêm os atributos que servem para definir os *namespaces* de um documento WSDL, este elemento raiz chama-se *definitions*.

Os *namespaces* são utilizados pelo WSDL, que aumentam a reutilização dos elementos/componentes definidos em seu documento. E Reckziegel (2006), descreve *namespaces* como sendo espaços para nomes que serão definidos no interior de um XML, o que permite a unicidade de nomes.

Dentre todos os *namespaces* existentes usados em um documento WSDL, no Quadro 2 é listado os principais.

Prefixo	**URI do *namespace***	**Descrição**
WSDL	http://schemas.xmlsoap.org/wsdl	*Namespace* de WSDL para Framework WSDL
HTTP	http://schemas.xmlsoap.org/wsdl/http	*Namespace* de WSDL para WSDL HTTP GET e vínculo POST
MIME	http://schemas.xmlsoap.org/wsdl/mime	*Namespace* de WSDL para vínculo MINE DE WSDL
XSD	http://www.w3.org/2001/XMLSchema	*Namespace* de esquema conforme definido pelo esquema XSD.

Quadro 2 – Prefixo de *namespaces* mais usados no WSDL
Fonte: Adaptado de Seely (2002)

Portanto o padrão WSDL define *Web Service* como uma coleção de portas, e estas permitem algumas operações que implicam na troca de algumas mensagens. E que são formadas por tipos de dados definidos em um schema XML (SAMPAIO, 2006).

2.2 *Web Service* REST (*Representational State Transfer*)

REST teve sua origem no ano de 2000, quando Roy Fielding escreveu sua tese de doutorado. Roy Fielding (é um dos principais autores da especificação HTTP), que descreve um estilo de arquitetura de *software* para sistemas hipermídias distribuídos.

Em sua tese Fielding (2000) diz que "Transferência de Estado Representacional" destina-se a evocar uma imagem de como um aplicativo da web bem projetado se comporta: uma rede de páginas web (um virtual Estado-máquina), onde o usuário progride através da aplicação selecionando ligações (transições de estado),

resultando na página seguinte (que representa o estado seguinte da aplicação) serem transferidos para o utilizador e processado para a sua utilização.

2.2.1 Compondo o REST

Vale ressaltar que REST não é um padrão ou tecnologia, apenas representa um modelo de como a Internet deve funcionar. Não existe nem mesmo uma entidade que efetue sua padronização.

Esse fato acontece porque, REST é um conjunto de regras que definem quais os conectores, recursos, componentes que podem ser usados para compor um sistema, conforme cita Fielding (2011) em sua dissertação.

2.2.2 Recursos

Em REST tem o que chamamos de recurso. Fielding (2000) descreve recurso como sendo tudo aquilo que pode ser nomeado. No livro *Restful Web Services*, eles são definidos como tudo que podemos criar um link para acesso.

Para uma melhor análise nos remeteremos a Filho e Ferreira (2009, p. 37;38) que definem recurso como “uma abstração ou conceito relevante que existe no domínio tratado pelo serviço em questão. Adicionalmente, um recurso pode compreender um grupo de objetos”.

Recursos estes que são reconhecidos por meio de suas URIs (*Uniform Resource Identifier*) únicas. Que podem ser representados de várias formas, como por exemplo, utilizando os formatos mais conhecidos e utilizados na web: XML e JSON (*Java Script Object Notation*).

2.2.3 Sem Estado

Outro princípio que chama atenção, que é bastante discutido do REST é o fato de ser uma aplicação sem estado (*stateless*). Onde não é exigido nenhum estado anterior do cliente no servidor. Uma requisição é suficiente para realizar determinada operação, e o servidor também não guarda nada dessa requisição.

Na visão de José (2009) este princípio de serviços sem estado também traz algumas desvantagens como é o fato “de que os dados precisam ser enviados a cada nova requisição”.

2.2.4 Interface Uniforme

Vale destacar que os recursos são todos acessados por meio de uma interface genérica o HTTP. Este protocolo oferece cinco métodos principais para acesso aos recursos: GET, HEAD, POST, PUT e DELETE.

De acordo com Bill (2009) o grande problema desta interface uniforme “é que toda a troca de informação é feita de uma forma genérica ao invés de ser específica para cada necessidade”. O autor ainda afirma que “REST foi projetado para ser eficiente na troca de dados hipermídia no caso, comum, a web”.

2.2.5 Recursos HTTP

O uso de recursos HTTP em REST é combinado através da manipulação de dados geralmente com as operações CRUD (*Create, Read, Update, e Delete*), conforme descrito no Quadro 3, pois assim é possível efetuar a persistência e recuperação dos dados.

Métodos HTTP	Operações CRUD
GET	SELECT
POST	INSERT, UPDATE, DELETE
PUT	CREATE, UPDATE
DELETE	DELETE

Quadro 3 – Associação dos métodos HTTP com operações CRUD
Fonte: Santos (2009)

2.3 REST e API JAX-RS

REST é representado em Java através da API JAX-RS, que vem da criação da especificação JSR311. Que permite com facilidade a criação de sistemas que utilizam esta arquitetura.

Bill (2009) exemplifica JAX-RS como sendo um modelo baseado em anotações, onde se é definido uma URI para o acesso ao recurso, que por sua vez define uma mapeamento entre uma URI e uma classe Java.

Como citado anteriormente JAX-RS utiliza-se de anotações, logo se pode destacar uma das suas anotações de maior importância, o @*Path*, que dirá quando uma classe poderá ser acessada como sendo um recurso.

Na Figura 2 é possível observar que o mapeamento URI é feito através da anotação @*Path* em que a classe FichamentoRest atende ao recurso através da URI “entidades.fichamento”, ao fazer acesso a URI “entidades.fichamento”, acessa-se a classe onde serão retornados todos os fichamentos.

```
@Stateless
@Path("entidades.fichamento")
public class FichamentoREST extends AbstractFacade<Fichamento> {
    @PersistenceContext(unitName = "WebServiceFichamentoRESTEasyPU")
    private EntityManager em;

    public FichamentoREST() {
        super(Fichamento.class);
    }
```

Figura 2: Uso da anotação @*Path* para mapeamento da URI
Fonte: Autoria própria

Outras anotações são usadas para acessar os recursos e manipular seus dados, no JAX-RS tem-se as anotações: @*Get*, @*Post*, @*Put, @Delete*, que indicam a qual tipo de requisição HTTP o recurso responderá, conforme é observado na Figura 3.

```
@POST
@Override
@Consumes({"application/xml", "application/json"})
public void create(Fichamento entity) {
    super.create(entity);
}

@PUT
@Override
@Consumes({"application/xml", "application/json"})
public void edit(Fichamento entity) {
    super.edit(entity);
}

@DELETE
@Path("{id}")
public void remove(@PathParam("id") Integer id) {
    super.remove(super.find(id));
}

@GET
@Path("{id}")
@Produces({"application/xml", "application/json"})
public Fichamento find(@PathParam("id") Integer id) {
    return super.find(id);
}
```

Figura 3: URI respondendo a uma requisição HTTP do tipo *POST, PUT, DELETE* e *GET*
Fonte: Autoria própria

Para saber qual o formato aceito durante o pedido e resposta ao servidor, são declaradas as anotações *@Produces* e *@Consumes* na declaração da classe ou método.

Quando o servidor devolve uma resposta de uma solicitação, a anotação *@Produces*, que tem suporte a todos os tipos MIME (*Multipurpose Internet Mail Extensions*) do protocolo HTTP, no caso ("*application/xml", "application/json", "text/plain", "text/html*") define o formato da resposta do servidor. Como caracteriza (José, 2009).

A Figura 4 mostra o formato de retorno do método *@GET* como sendo dois tipos de retorno: "*application/xml", "application/json*".

```
@GET
@Path("{id}")
@Produces({"application/xml", "application/json"})
public Fichamento find(@PathParam("id") Integer id) {
    return super.find(id);
}
```

Figura 4: Tipos de response com a utilização da anotação *@Produces*
Fonte: Autoria própria

Já com a anotação *@Consumes* é definido quais os formatos que o recurso irá consumir, seguindo os mesmos padrões aplicados à anotação *@Produces*. Conforme se observa na Figura 5 o método *edit* consome apenas o formato XML.

```
@PUT
@Override
@Consumes({"application/xml"})
public void edit(Fichamento entity) {
    super.edit(entity);
}
```

Figura 5: Tipos de request com a utilização da anotação *@Consumes*
Fonte: Autoria própria

Além das anotações citadas anteriormente, também há uma série de outras anotações: que são brevemente mostradas abaixo:

- ***@XmlRootElement*:** Mapeia a classe para um elemento XML para que ela seja identificada pelo RESTEasy e então ser serializada para o formato XML ou JSON no corpo das requisições HTTP;
- **@*PathParam*:** auxilia na extração de informações de um requisição HTTP;
- ***@FormParam*:** capturar os valores submetidos a um formulário HTML (*HyperText Markup Language*), e assim transporta-los para parâmetros desejados;
- ***@QueryParam*:** utilizada para extrair o valor de um parâmetro indicado na *query*, e atribui-lo a um parâmetro ou atributo da classe;

- **@*HeaderParam*:** utilizada para extrair valores header HTTP, e associa-los a um parâmetro ou método, o qual pode ter diversas aplicações, como verificar qual browser está sendo utilizado.

2.4 Desenvolvimento de *Software*

2.4.1 Dificuldades

Como vimos anteriormente, a utilização da tecnologia *Web Services* envolve o conhecimento de diversos padrões. Costa (2008) alega que “frequentemente, construir mensagens SOAP e arquivos de descrição WSDL será uma tarefa “demorada” e suscetível a erros”.

Na visão de Prescod (2012) a dificuldade em desenvolver o acesso aos *Web Services* através do protocolo SOAP representa também um fator limitador para a disseminação do protocolo. As reflexões deste autor ainda apontam que atualmente, desenvolvedores estão acostumados a utilizar aplicativos que geram clientes automaticamente para acessar os serviços.

Porém os códigos fontes gerados para esses clientes possuem uma complexidade grande, e se acontecer uma mudança de interface, praticamente torna-se inviável alterar o cliente, sendo mais simples criar outro.

2.4.2 Facilidade e Segurança

O SOAP é baseado em XML, e utiliza esta linguagem de marcação, pela simplicidade e permite que o protocolo seja facilmente extensível (SUDA, 2003).

Outro ponto interessante do SOAP é que se houver uma combinação com as especificações WS-* podem cobrir as questões de QoS (*Quality of Service*), ou seja segurança nas transações de dados.

Pelo fato de uma mensagem SOAP poder ser propagada por diferentes protocolos, isso possibilita uma flexibilidade entre várias integrações.

2.4.3 Alternativa ao SOAP

Fielding (2000) definiu em sua famosa tese uma arquitetura que rapidamente se tornou uma alternativa ao SOAP. Apesar de não ser um protocolo, o REST segue os princípios dominantes na web para expor *Web Services*.

2.4.4 Facilidade e Vantagem

Web Services baseado em REST utiliza e também implementa alguns dos padrões utilizados na web, que são eles: XML, JSON, HTTP dentre outros. Desta forma, para desenvolver as aplicações clientes a tarefa se torna simples, pois as ferramentas de desenvolvimento possuem grande suporte para estes padrões.

Tyagi (2006) enfatiza que REST é recomendado para aplicações com recursos limitados, como exemplo aparelhos PDA (*Personal Digital Assistent*) e celulares, simplesmente por ter um baixo custo de comunicação.

A interface uniforme é um elemento fundamental para a implementação em RESTEasy. Pois o cliente HTTP é capaz de se comunicar com qualquer servidor HTTP sem detalhes da configuração.

2.4.5 Desvantagem

Uma grande desvantagem conforme Tyagi (2006) comenta em sua obra, é que REST não possui um padrão oficial para descrição dos seus serviços, até existe uma frente interessada em criar esse padrão utilizando WADL (*Web Application Description Language*), mas ainda não existe nada oficial.

Assim é possível observar que *Web Service*s implementados utilizando a arquitetura REST, não tendo um padrão oficial para definir a interface dos seus serviços, dificulta a maneira para publicar os serviços e também que seja feita uma descoberta dos mesmos.

3 DESENVOLVIMENTO

3.1 Implementação SOAP

Para desenvolver o *Web Service* SOAP foi utilizado a plataforma de programação para servidores na linguagem de programação Java. Escolhido então o servidor de aplicações *GlassFish Server* 3.1.2, não precisou baixar externamente, pois já vem embutido na IDE Netbeans, na versão 7.2.

O projeto intitulado *WebService*FichamentoSOAP, será o servidor. Dando continuidade, foi criado uma classe de entidade, para representar a tabela *webservice* do banco de dados MYSQL, como uma classe Java. No momento da criação desta classe de entidade do banco de dados é solicitado criar uma conexão com o banco para poder acessar seus dados. Sendo assim foi selecionado o driver MySQL (Conector/J Driver) passado junto o nome da base de dados, usuário e senha do banco.

Foi criado então a classe Fichamento.Java dentro do pacote org.ws.fichamento.modelo. O código ilustrado na Figura 6 representa como foi feita a representação do banco como uma classe.

```
public class Fichamento implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "idFichamento")
    private Integer idFichamento;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 40)
    @Column(name = "curso")
    private String curso;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 255)
```

Figura 6 – Trecho do Código Fonte da classe de entidade
Fonte: Autoria própria

Dando continuidade foi criado as classes de interface local e remota. A classe FichamentoFacadeRemote terá os métodos de listar os fichamentos, no geral ou por período, conforme mostra a Figura 7.

```
package org.ws.fichamento.modelo;
import java.util.List;
import javax.ejb.Remote;
@Remote
public interface FichamentoFacadeRemote {
        public List<Fichamento> listarFichamentoPorPeriodo(String datapublicacao1, String datapublicacao2);
        public List<Fichamento> listarFichamento();
```

Figura 7 – Classe Remota contendo os métodos de listar fichamentos
Fonte: Autoria própria

Já a classe FichamentoFacade (local) terá descritos os métodos em si, que chama as funções da classe Fichamento que contém os get´s e os endereços de referência ao banco de dados. Segue exemplo do código na Figura 8.

```
public List<Fichamento> listarFichamento() {
        return em.createNamedQuery("Fichamento.findAll").getResultList();
}
```

Figura 8 – Método listar fichamentos
Fonte: Autoria própria

Agora chegou o momento de criar o *Web Service* que ficará dentro do pacote org.ws.fichamento.webservice. Neste *Web Service* conterá os *@WebMethod*, conforme ilustra a Figura 9, que invoca os métodos listarFichamento e listarFichamentoPorPeriodo demostrado na Figura 8.

```
    // Retorna a lista de todos os registros da tabela "fichamento"
    @WebMethod(operationName = "listarFichamento")
    public List<Fichamento> listarFichamento() {
            return fichamentoFacadeBean.listarFichamento();
    }
```

```
// Retorna a lista de todos os registros da tabela "fichamento",
// que estejam dentro do período
@WebMethod(operationName = "listarFichamentoPorPeriodo")
public List<Fichamento> listarFichamentoPorPeriodo(
        @WebParam(name = "datapublicacao1") String datapublicacao1,
        @WebParam(name = "datapublicacao2") String datapublicacao2) {
        return fichamentoFacadeBean.listarFichamentoPorPeriodo(datapublicacao1
}
```

Figura 9 – *@WebMethod* dos métodos listar fichamentos
Fonte: Autoria própria

Neste momento já é possível testar o *Web Service*, fazendo sua construção (*Build*) e depois sua implantação (*Deploy*). Ao fazer o *Deploy* o servidor de aplicações *GlassFish* se inicia automaticamente. Abaixo na Figura 10, a página web do *Web Service*.

Figura 10 – Página de teste do *Web Service*
Fonte: Autoria própria

Na ilustração da Figura 11 é possível visualizar a estrutura do projeto, e logo após na Figura 12, tem-se o diagrama de classe UML do *Web Service* SOAP.

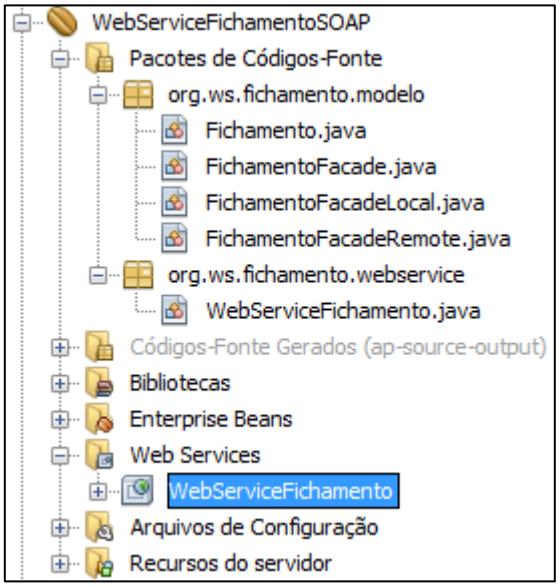

Figura 11 – Estrutura do projeto EJB
Fonte: Autoria própria

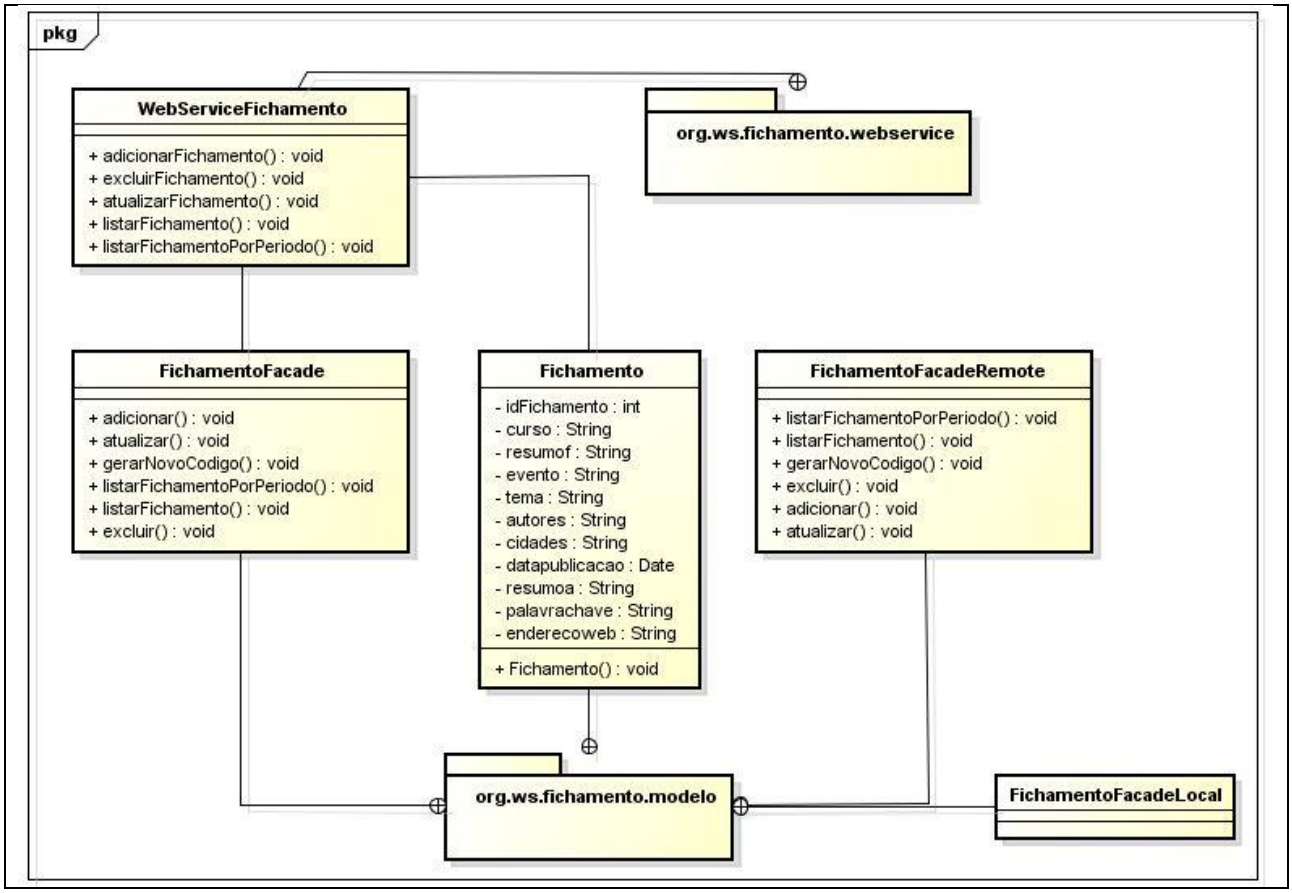

Figura 12 – Diagrama UML *Web Service* SOAP
Fonte: Autoria própria

3.2 Implementação RESTEasy

No desenvolvimento do *Web Service* baseado na arquitetura REST, foi necessário efetuar o *download* dos arquivos .jar da implementação RESTEasy. O servidor de aplicações utilizado foi o *GlassFish*.

Após criar um novo projeto do tipo Java Web, com o nome WebServiceFichamentoREStEasy, foram adicionados todos os .jar da biblioteca RESTEasy dentro da pasta do projeto ../WEB-INF/LIB. Fazendo isso o Netbeans reconheceu todas as classes ao RESTEasy.

Para fazer a conexão do banco de dados e fazer com que ele se representasse como uma classe foi criado no projeto um *Web Service* RESTFul a partir do banco de

dados. Aqui da mesma forma que aconteceu no SOAP também é especificado o driver MYSQL, o nome do banco, usuário e senha.

Agora é necessário declarar a classe de recurso no descritor da aplicação que fica em web.xml. O conteúdo do arquivo ficou da seguinte maneira, conforme a Figura 13.

```
<display-name>Restful Web Application</display-name>

<!-- REST Resource -->
<context-param>
   <param-name>resteasy.resources</param-name>
   <param-value>br.com.bc.rest.service.LibraryResource</param-value>
</context-param>

<listener>
   <listener-class>org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap</listener-class>
</listener>

<servlet>
   <servlet-name>resteasy-servlet</servlet-name>
   <servlet-class>org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>resteasy-servlet</servlet-name>
   <url-pattern>/*</url-pattern>
</servlet-mapping>
```

Figura 13 – Descritor da aplicação web.xml
Fonte: Autoria própria

Com isso já é possível construir o pacote para fazer o *deploy*, o resultado do *Web Service* RESTEasy é mostrado abaixo na Figura 14:

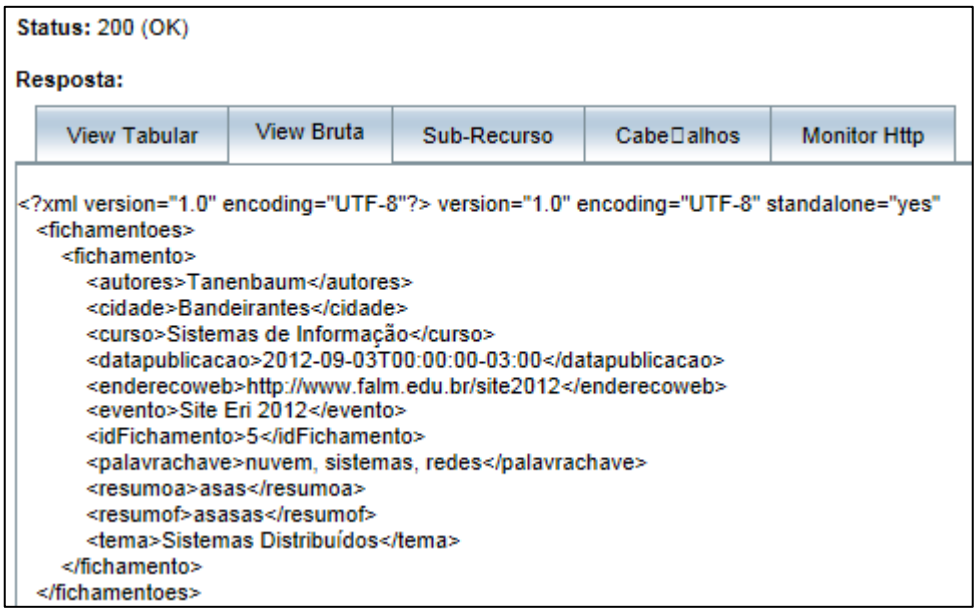

Figura 14 – Resultado do *Web Service* RESTEasy
Fonte: Autoria própria

É possível notar na Figura 15, os detalhes da página RESTEasy. No topo o endereço do WADL, tem a localização do recurso, e os métodos para efetuar os testes.

Figura 15 – Página do Test RESTFul Web Services
Fonte: Autoria própria

Através da Figura 16, é possível visualizar a estrutura do diagrama de classes UML do projeto *Web Service* RESTEasy.

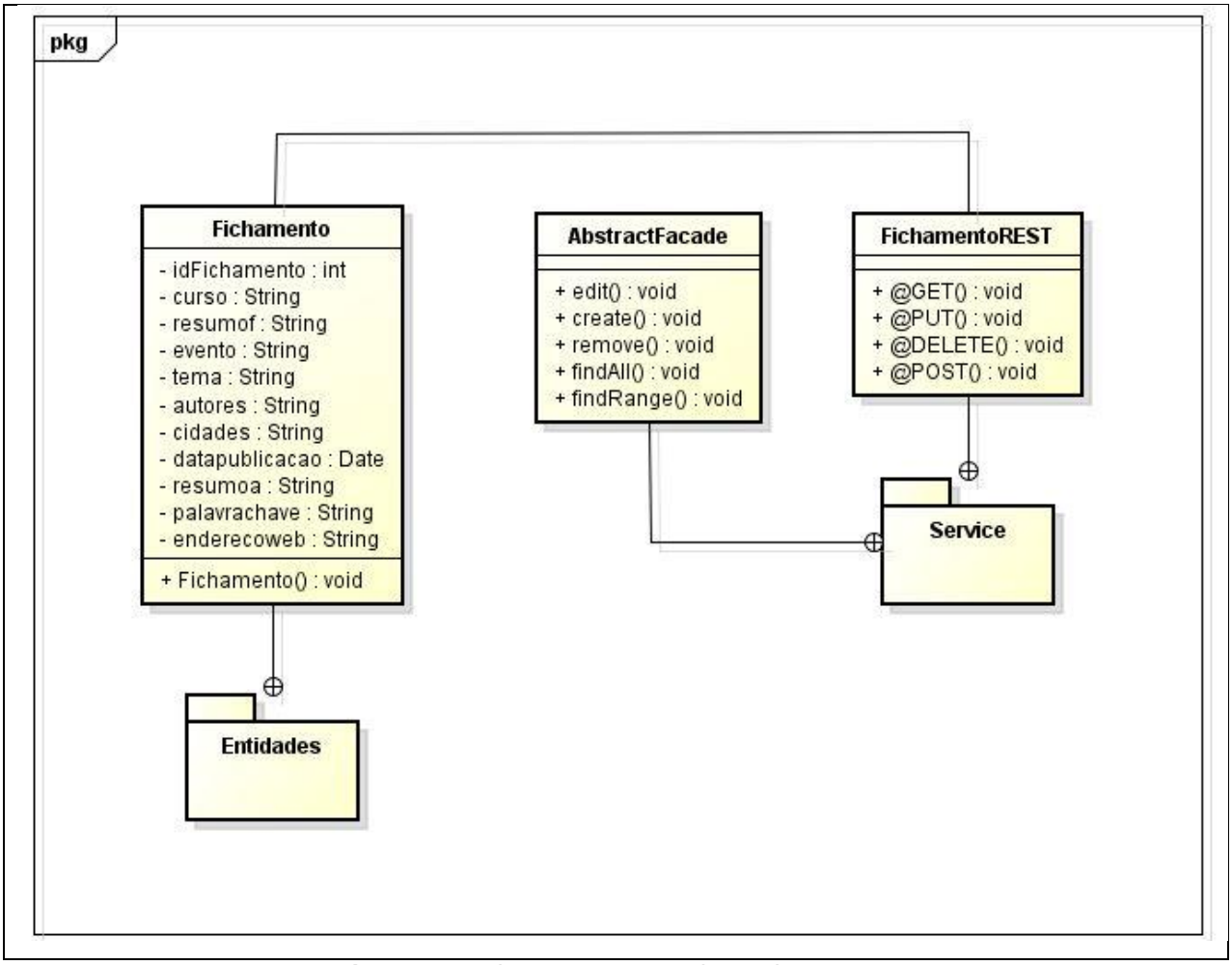

Figura 16 – Diagrama UML *Web Service* REST
Fonte: Autoria própria

A implementação dos *Web Services* foram efetuados com sucesso, abaixo é apresentado à ferramenta de teste utilizada. E na Figura 17, a estrutura do banco de dados, representado através do diagrama entidade relacionamento.

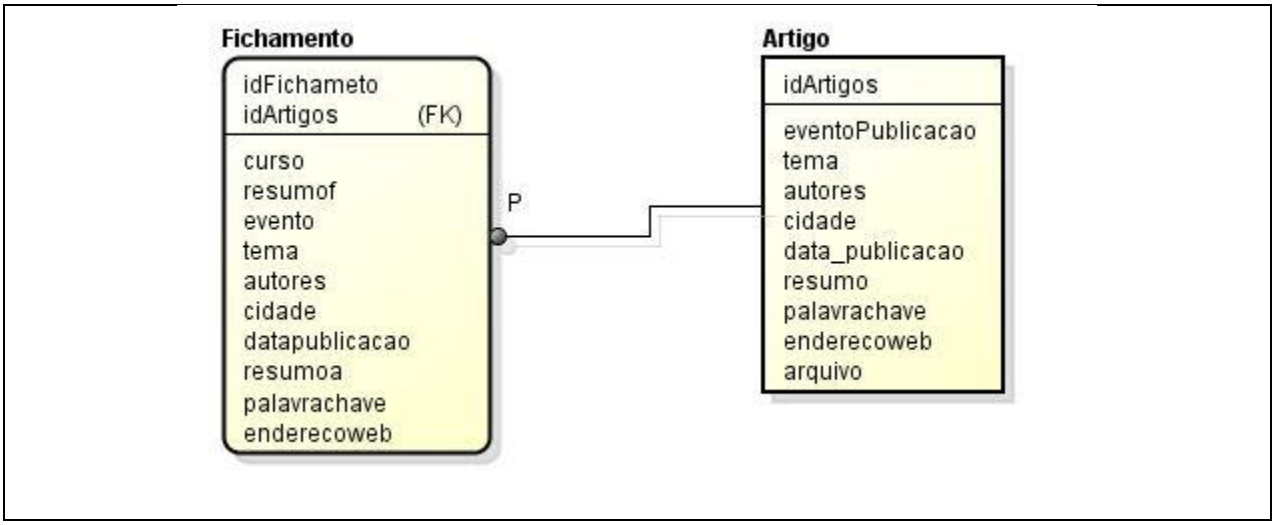

Figura 17 – Modelo Entidade Relacionamento Banco de Dados
Fonte: Autoria própria

3.3 JMeter

O JMeter foi escolhido para ser a ferramenta responsável pelos testes de carga, pois é uma ferramenta *Open Source* de testes de desempenho automatizados para aplicações web. O JMeter oferece recursos para realizar testes de desempenho, volume e estresse automatizados para aplicações web, servidores FTP, *Web Services*, banco de dados, entre outros.

3.3.1 Conhecendo a ferramenta de teste

É uma ferramenta que simula a utilização de software por meio de usuários virtuais, simula vários usuários acessando o sistema ao mesmo tempo.

Uma das suas características é por ser multiplataforma, multithreading, conter gráficos e possuir interface gráfica. Simplificar a procura por *bugs* e o desempenho da aplicação são alguns dos fatores que levam a utilização do JMeter.

JMeter é a ferramenta de automação apropriada para o tipo de teste de desempenho e estresse (SOUZA E ALCÂNTARA, 2008).

Stantchev (2009), também comenta sobre o JMeter em seu artigo, dizendo que há uma variedade de ferramentas que podem servir como condutores de teste. Exemplos são *Load Runner* da HP, a extensão do *Rational Performance Tester* para

SOA *Quality* da IBM, o JMeter Apache, e o Teste de Arquitetura *Open System* (OpenSTA).

Logo abaixo é apresentado a ferramenta JMeter e a forma que foi criado o teste de desempenho em ambos *Web Services*.

1 - Com o JMeter aberto, como mostra a Figura 18, tem que ser criado o “script” para o teste.

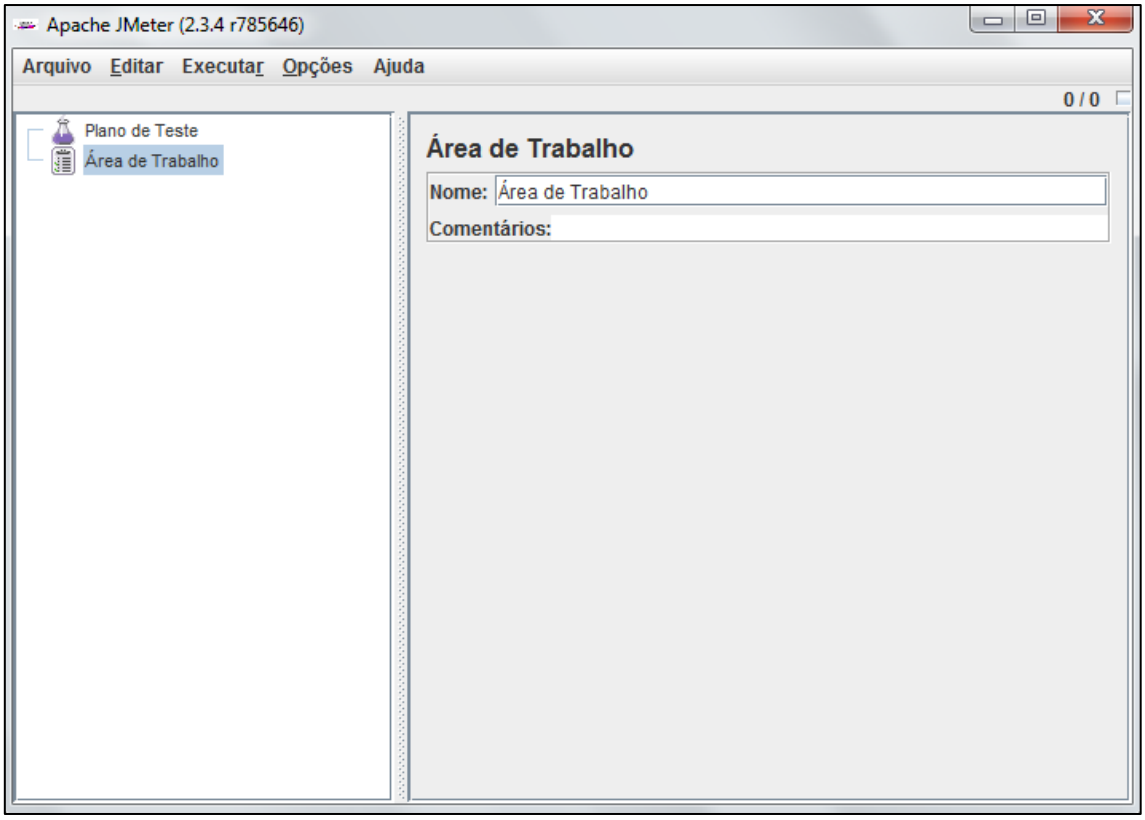

Figura 18 – Tela Inicial do JMeter
Fonte: Autoria própria

2 – Para tal tarefa tem que ser adicionado:

- O componente principal: “Grupo de Usuários”;
- Caminho: botão direito do mouse sobre o item “Plano de Teste”, em seguida selecionado “Adicionar”, e para finalizar, “Grupo de usuários”, conforme ilustra a Figura 19.

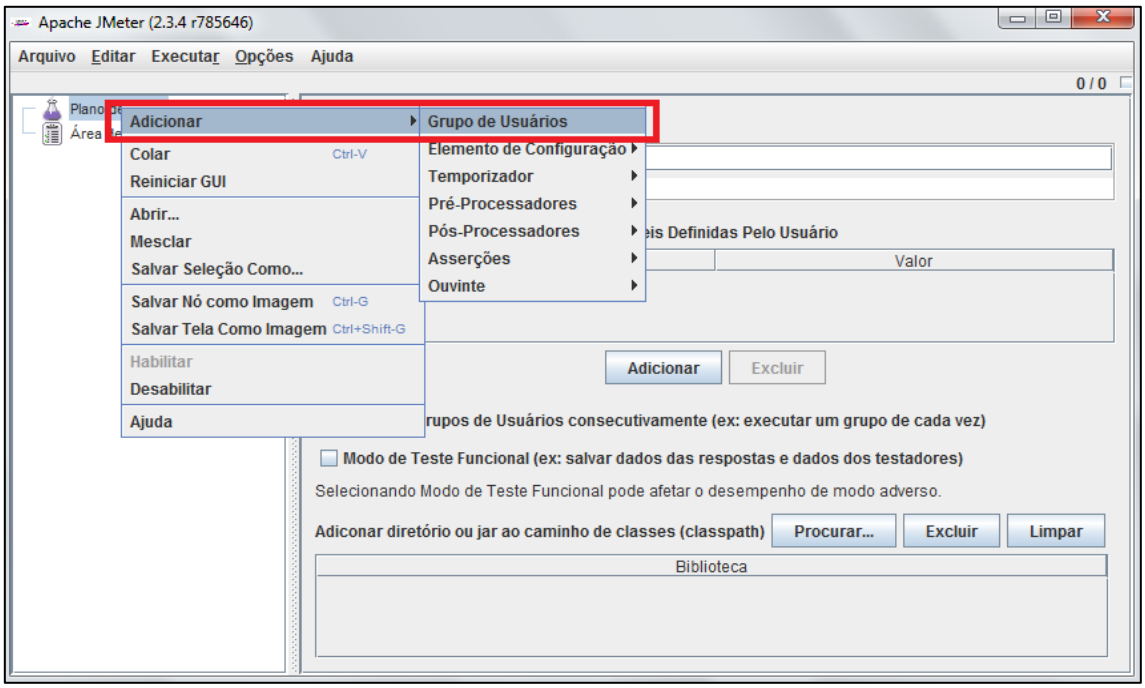

Figura 19 – Criando o Grupo de Usuários
Fonte: Autoria própria

3 - Configurando o “Grupo de usuários”:

Observações dos campos:

- **Nome:** Grupo de Usuários Teste RESTEasy_SOAP;
- **Número de usuários virtuais (threads):** Número de usuários simultâneos que deseja simular. No caso, foram solicitados 2000 usuários simultâneos, então, neste campo o número informado foi 2000;
- **Tempo de inicialização (em segundos):** Tempo em que os usuários virtuais (threads) serão inicializados. Por exemplo, se informar no campo acima 2000 *threads*, e o tempo de inicialização estiver como "1", as 2000 *threads* (usuários), serão inicializadas a cada 1 segundo;
- **Contador de Iteração:** Quantas vezes o teste será executado. Por exemplo, para simular 2000 usuários simultâneos acessando a página de busca do Google, 6 vezes. Então, neste campo será informado o número 6.

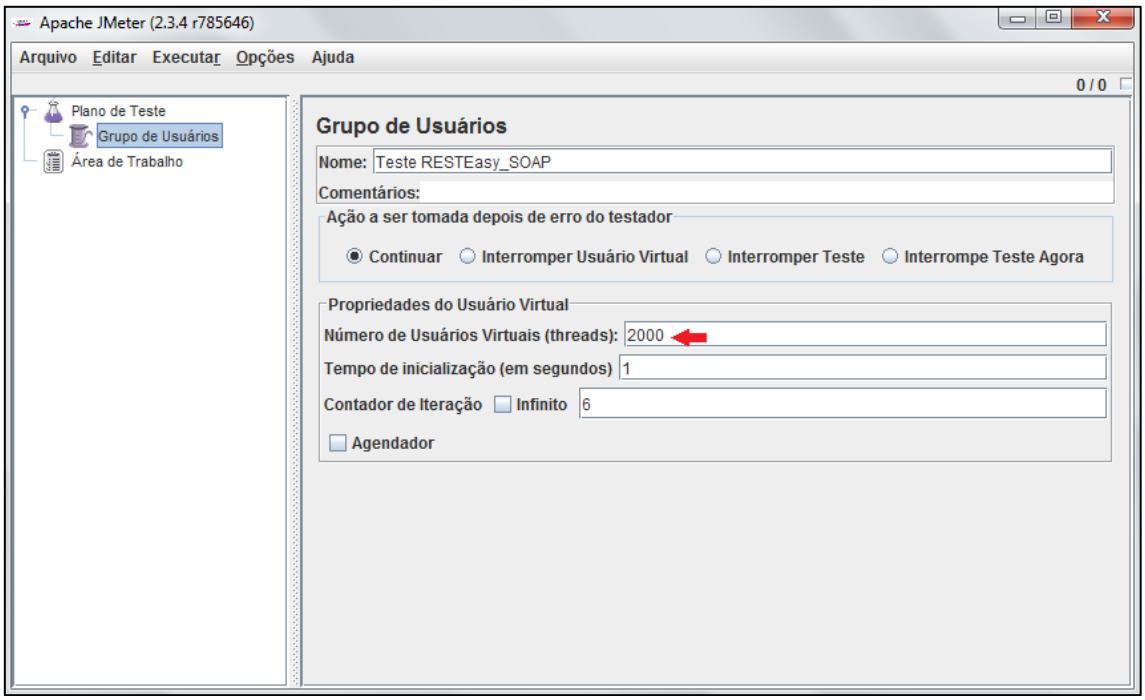

Figura 20 – Configurando o Grupo de Usuários
Fonte: Autoria própria

4 - O próximo passo foi adicionar um novo componente, dentro de “Grupo de usuários”, que foi renomeado para “Teste RESTEasy_SOAP”.

- Caminho: botão direito sobre o “Teste RESTEasy_SOAP”, foi selecionado a opção “Adicionar”, em seguida “Testador”, e por fim “Requisição HTTP, ou Requisição SOAP”, conforme ilustra a Figura 21.

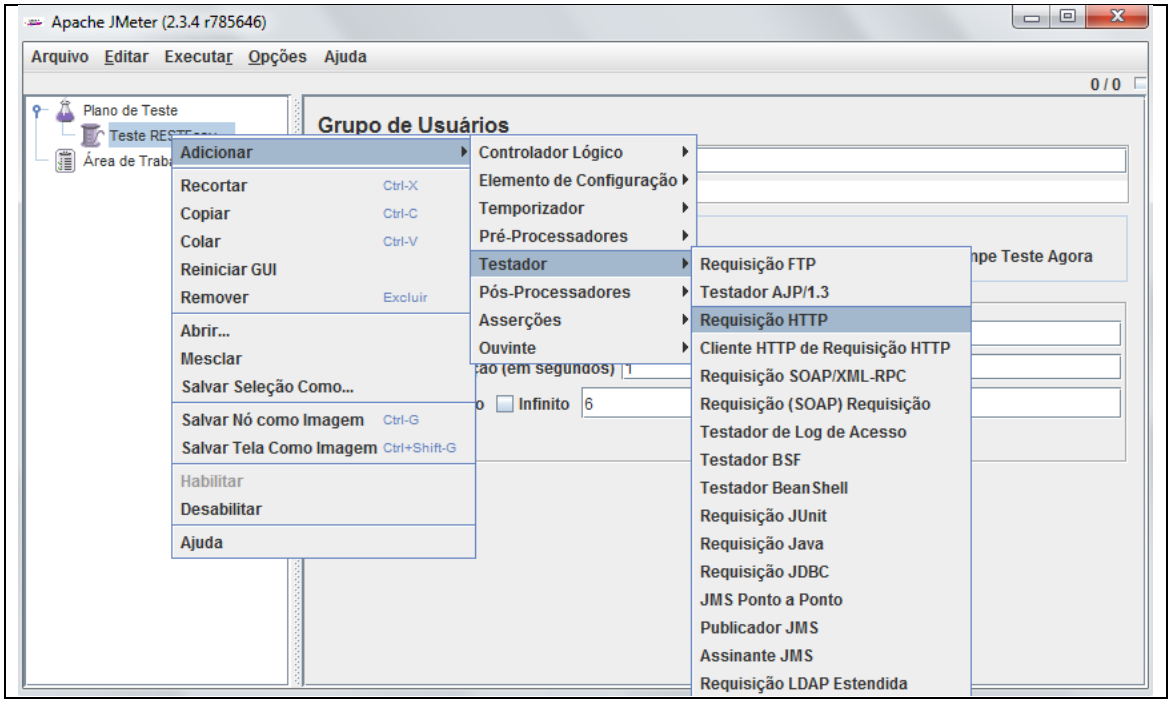

Figura 21 – Adicionando um testador no JMeter
Fonte: Autoria própria

5 - Agora, tem que ser configurado o componente “Requisição HTTP” que foi adicionado, conforme a Figura 22.

- **Nome:** Requisição HTTP RESTEasy;
- **Comentários:** Requisições efetuados no servidor RESTEasy;
- **Protocolo padrão (HTTP):** Como a página não é autenticada (HTTPS) foi usado o padrão (HTTP), portanto foi informado o padrão no campo;
- **Caminho:** http://localhost:8080/WebServiceFichamentoRESTEasy/webresources/entidades.fichamento?#.

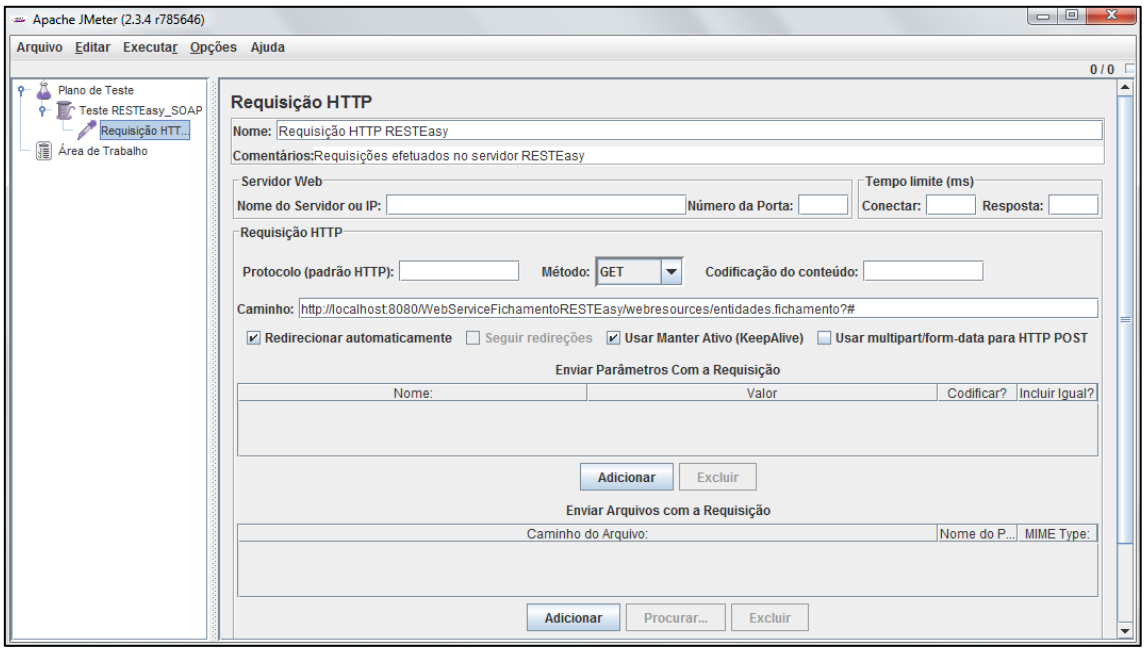

Figura 22 – Configurado o testador HTTP
Fonte: Autoria própria

6 – Até o presente momento, o script já poderia ser executado, entretanto, não seria possível visualizar nenhum resultado. Para visualizar o resultado do script, tem que ser adicionado um componente muito importante que é o "Ouvinte". Um componente do tipo "Ouvinte" tem como finalidade exibir os resultados dos scripts. É importante destacar que existem diversos tipos, mais o utilizado no presente trabalho foi "Ver árvore de resultados".

Para adicionar este componente, foi acionado o botão direito do mouse sobre a Requisição HTTP, selecionado a opção "Adicionar", "Ouvinte" e em seguida, a opção "Ver árvore de resultados", conforme a Figura 23.

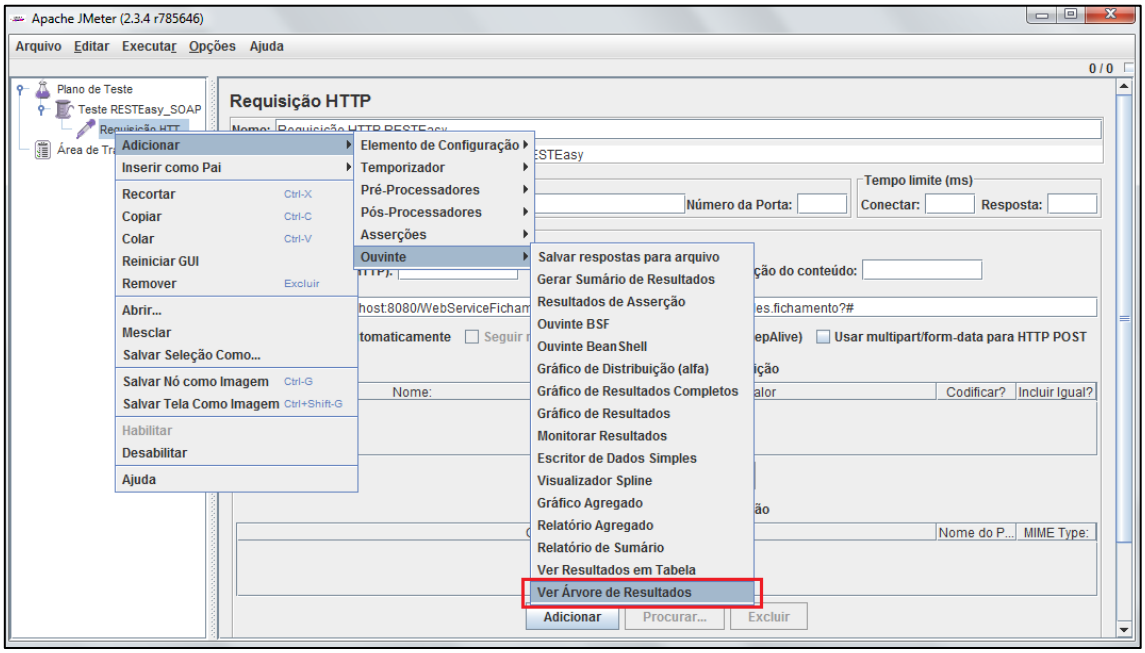

Figura 23 – Adicionando o ouvinte Árvore de Resultados
Fonte: Autoria própria

7 – Por fim basta executar o script, para isso, ir ao menu "Executar" e em seguida a opção "Iniciar", conforme ilustra a Figura 24:

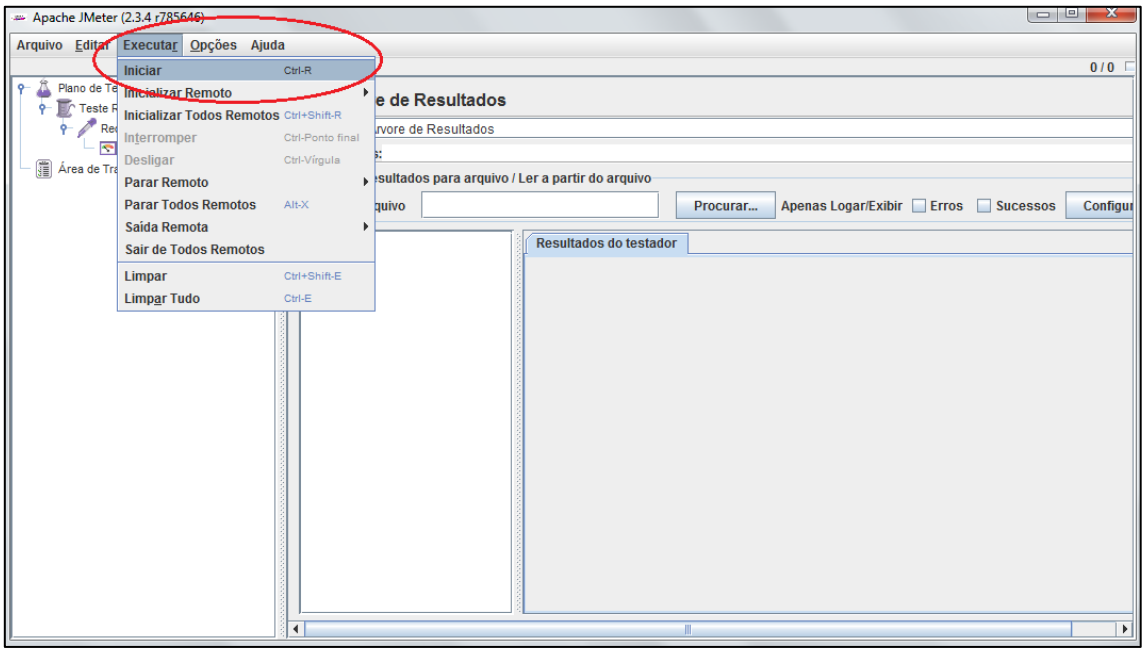

Figura 24 – Iniciando o teste de desempenho
Fonte: Autoria própria

8 - Para verificar se o script está sendo executado, é só observar um quadradinho no canto superior direito, Figura 25, ele ficará verde quando estiver em execução, e "vazio", quando finalizado a execução.

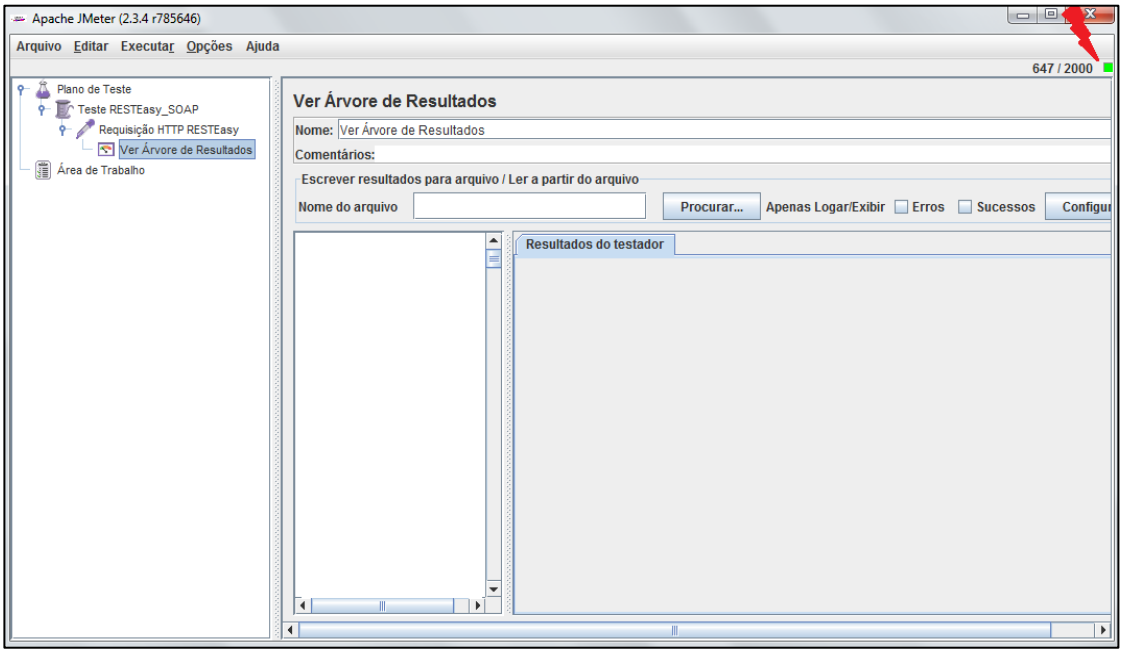

Figura 25 – Script sendo executado
Fonte: Autoria própria

4 Resultados Obtidos

4.1 Teste de carga com JMeter

O servidor utilizado para realizar os testes é equipado com um processador *Core* I5, contendo 4Gb de Memória *Ram*, 500Gb de HD, e rodando o Windows 7 64Bits.

Já o cliente possui as seguintes configurações: processador Turion X2, 2GB de Memória *Ram*, 500Gb de HD, e rodando o Windows 7 64Bits.

A métrica utilizada para tal tarefa foi o número de requisições que os *Web Services* iria suportar sem apresentar erros, e a velocidade de respostas das requisições.

Para a realização dos testes foi configurado a quantidade de 2.000 (dois mil) usuários virtuais, ou seja, as *threads*. Esta foi a quantidade suportada pelo *Hardware*, acima deste valor a máquina trava em algum momento do teste.

O tempo de inicialização de cada *thread* foi definido como sendo 1 (um) segundos por teste.

E as repetições ficaram definidas em 6 (seis) vezes, ou seja, os dois mil *threads* seriam executados 6 (seis) vezes sem pausas, fazendo com isso que 12000 (doze mil) amostras fossem executadas no teste.

Os testes de carga foram repetidos 30 (trinta) vezes, e foram testados em 3 (três) ambientes de redes diferentes, que são:

- Utilizando a própria máquina para ser Servidor e Cliente, ou seja, *Local host*;
- Conectando duas máquinas via cada de rede, ficando assim o acesso local;
- E por fim, testado em rede, onde se tem um *Switch* fazendo o gerenciamento da rede.

4.2 Análise da coleta dos dados

- Análise sobre o teste via Local host.

Através da análise realizada, constatou-se que o *Web Service* em RESTEasy só irá parar de apresentar erros, quando o tempo de inicialização dos *threads* for superior a 4 (quatro) segundos, ou, limitando a quantidade de amostras para 2.000 (dois mil).

Nota-se que o *Web Service* em SOAP não apresentou erros no seu teste de carga, usando a mesma configuração do *Web Service* RESTEasy.

Abaixo no Gráfico 1 é possível observar as requisições efetuadas durante os 30 testes e avaliar os dados obtidos no teste de carga dos *Web Service*s.

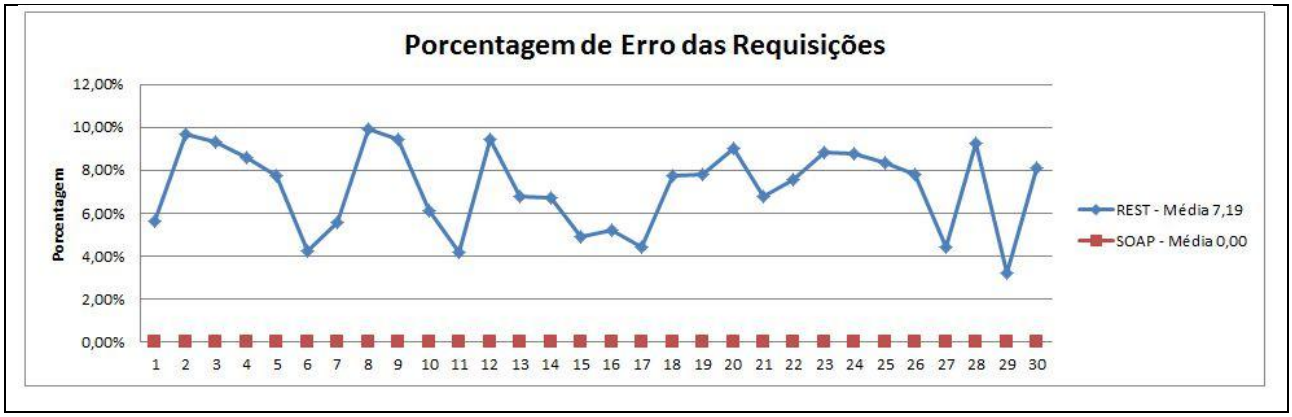

Gráfico 1 – Requisições via Local host
Fonte: Autoria própria

Já na ilustração do Gráfico 2, é possível observar a média ao longo da execução dos testes de carga e também a média geral das requisições em milissegundos.

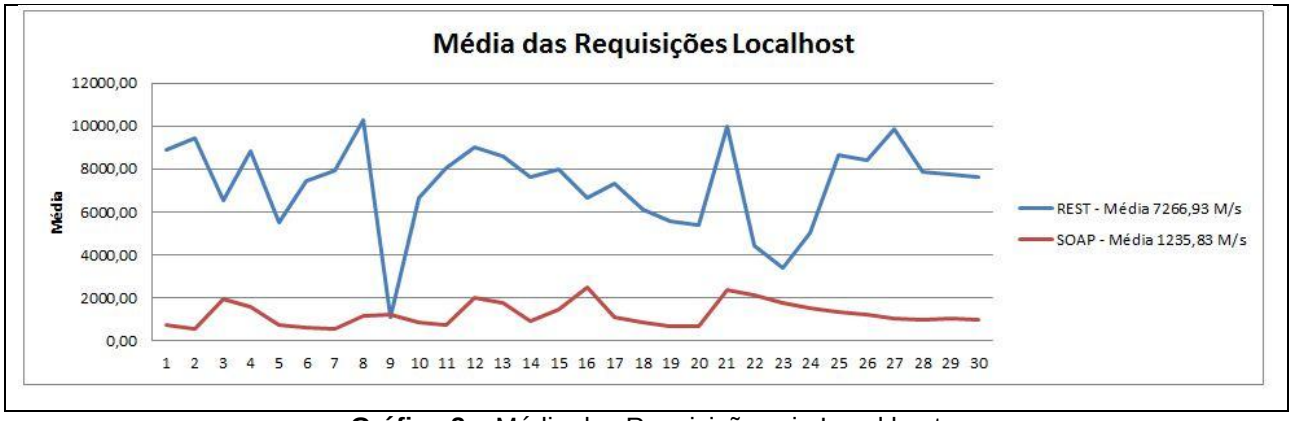

Gráfico 2 – Média das Requisições via Local host
Fonte: Autoria própria

- Análise sobre o teste via conexão local.

Nos resultados obtidos com a conexão local entre dois computadores a uma inversão de valores, pois a eficiência do *Web Service* SOAP via Local host não é vista neste cenário. O que se pode notar é uma grande estabilidade por parte do *Web Service* em RESTEasy e uma grande instabilidade por parte do serviço web em SOAP. Conforme o Gráfico 3 demostra com os dados das requisições.

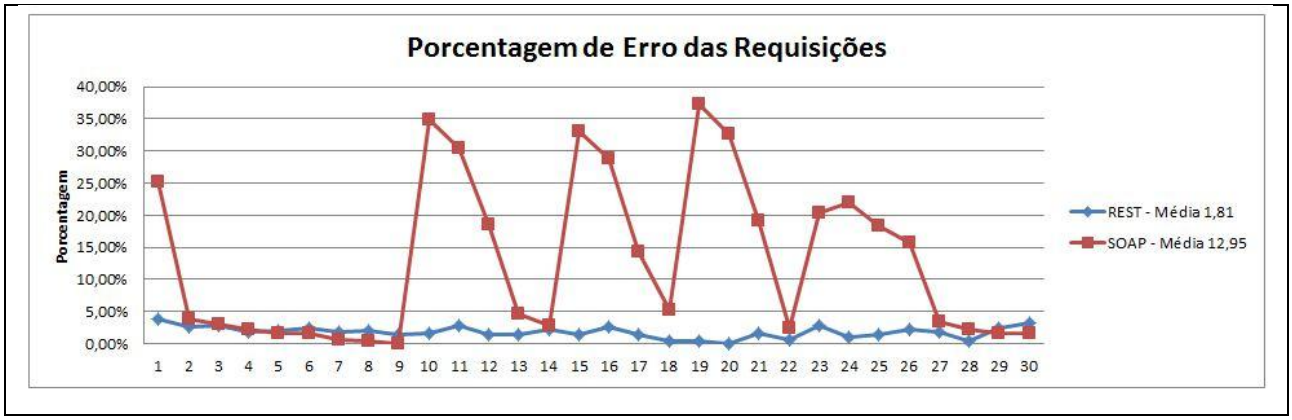

Gráfico 3 – Requisições via conexão local
Fonte: Autoria própria

A mesma coisa pode ser observada nas médias obtidas das requisições em ambos *Web Services*, conforme se vê no Gráfico 4.

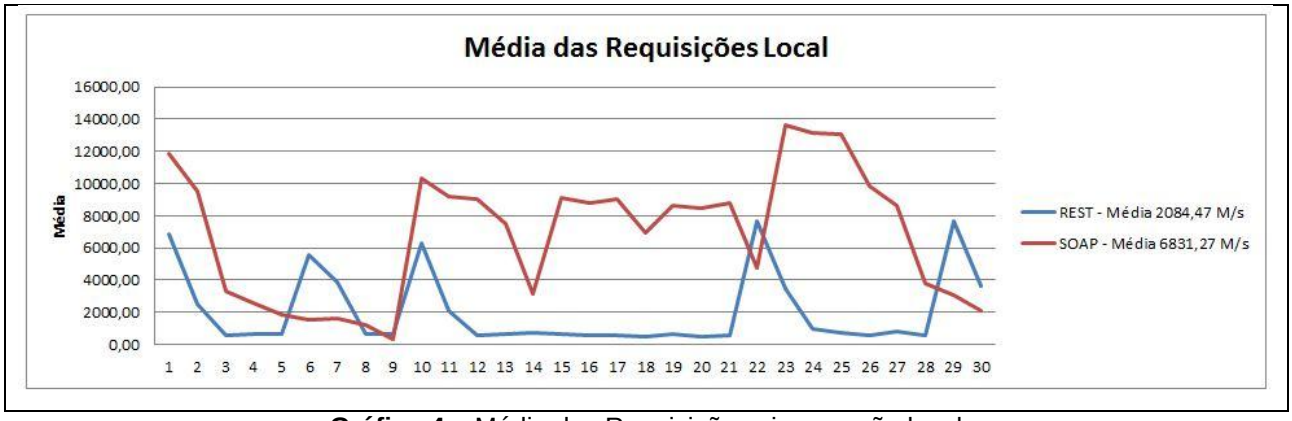

Gráfico 4 – Média das Requisições via conexão local
Fonte: Autoria própria

- Análise sobre o teste via Rede.

Nos testes realizados via rede, os resultados são semelhantes ao teste via conexão local, Gráfico 5, porém as médias das requisições são superiores, Gráfico 6, este fato ocorre por se tratar de uma rede que tem o tráfego gerenciado por um Switch.

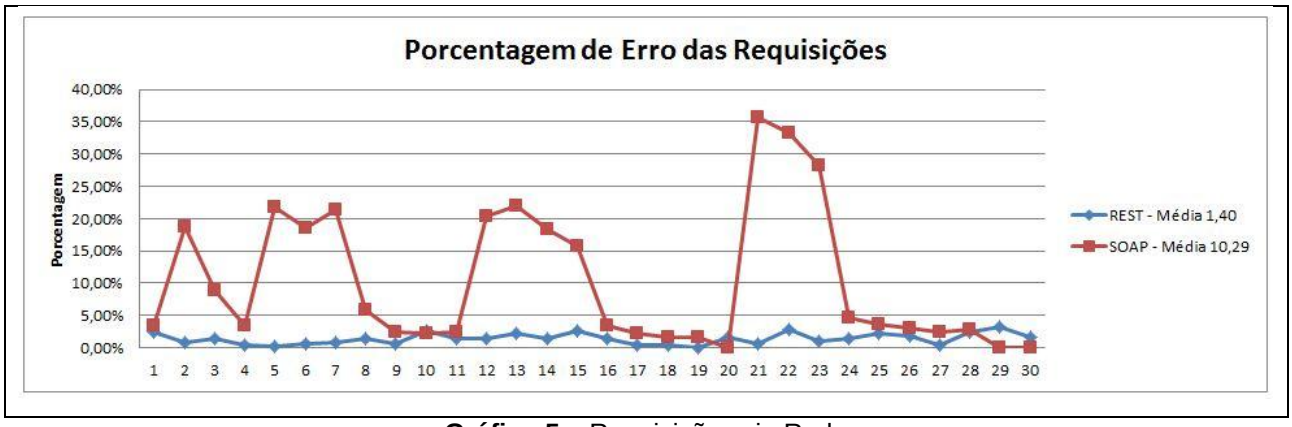

Gráfico 5 – Requisições via Rede
Fonte: Autoria própria

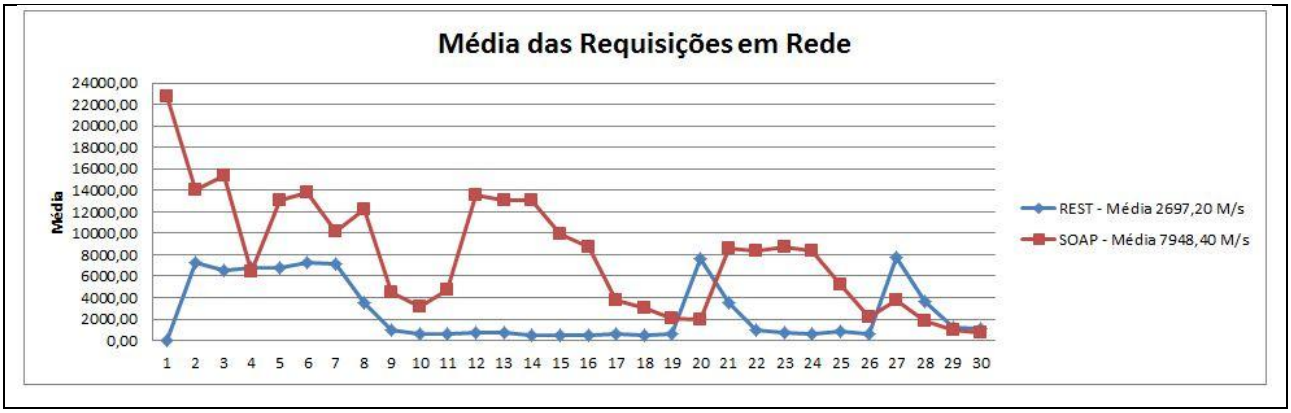

Gráfico 6 – Média das Requisições via Rede
Fonte: Autoria própria

4.3 Comparação entre códigos

Ao se comparar as linhas de códigos das implementações chega-se a conclusão que somado os fatores, quantidade de classes, códigos e tamanho do projeto, o *Web Service* REST implementado utilizando o RESTEasy é menor.

Abaixo na Tabela 1, podem-se conferir as diferenças entre estas arquiteturas.

	SOAP	REST
Classes	5	3
Linhas de Código	418	372
Tamanho do Projeto	276 Kb	232 Kb

Tabela 1 – Comparação entre linhas de código
Fonte: Autoria própria

5 CONCLUSÃO

O presente trabalho procurou abordar uma visão geral sobre *Web Services* SOAP e REST e principalmente o uso do *Web Service* baseado na arquitetura REST em Java, no qual foi implementado utilizando a biblioteca RESTEasy, proporcionando desde conhecimentos iniciais aos *Web Services*, para que o leitor possa entender como funciona este recurso, e despertar o interesse de estudo do mesmo.

Conseguiu-se alcançar todos os objetivos descritos para o trabalho em questão, que foi criar dois *Web Services* utilizando arquiteturas diferentes com interface para utilização e realizado o teste de carga com a ferramenta JMeter.

Foram exemplificados alguns trechos de códigos que detalham a diferença dos *Web Services*, mas há que se ressaltar que existem outras maneiras de implementação dos mesmos, portanto não se pode considerar a única ou a melhor forma de implementação, mas sim uma das formas possíveis de se implementar SOAP e REST em Java, tendo em vista que a melhor forma é aquela que atende as necessidades do desenvolvedor para aquela aplicação em específico.

É possível afirmar também que a biblioteca RESTEasy utilizada para implementar o *Web Service* REST ajuda no desenvolvimento pois é relativamente simples sua utilização, já que conta com a exploração de recursos HTTP.

E ainda atende as necessidades primárias de um *Web Service* normal, que é a comunicação de aplicações através da Internet. Por ser mais leve, é viável também utilizar REST em aplicativos móveis como celulares e PDA´s. Simplesmente por ter um baixo *overhead*, ou seja, um limite ou excesso na comunicação (TYAGI, 2006).

Após a realização dos testes de carga realizado com a ferramenta JMeter pode se notar que a simplicidade da implementação em RESTEasy é vantajosa para alguém que esteja preocupado com o desempenho de sua aplicação, pois nos resultados obtidos o RESTEasy tem uma eficiência maior utilizando uma rede LAN ou uma conexão local entre computadores. O seu único desempenho inferior ao *Web Service* SOAP ficou por conta da utilização via Local host, onde se tem o cliente e servidor na mesma máquina.

Com isso conclui-se que o *Web Service* em RESTEasy além de ter sua implementação mais fácil, possuir menos códigos e classes que o *Web Service* SOAP também é mais eficiente e tem uma estabilidade maior.

REFERÊNCIAS BIBLIOGRÁFICAS

ABILIO, JORGE; DUEIRE, RAFAEL. ***Web Services* em Java**. Rio de Janeiro: Brasport, 2006.

BURKE, Bill. ***RESTFul Java with JAX-RS**. United States of America: O´Relly Media*, 2010.

COSTA, Daniel Gouveia. **Java em rede, recursos avançados de programação**. Rio de Janeiro: Brasport, 2008.

DEITEL, HARVEY M. **XML como programar**. Porto Alegre: Artmed, 2001.

FARIA, R. A. **Treinamento avançado em XML: desvende os poderosos recursos desta linguagem**. São Paulo: Editora Digerati Books, 2005.

FIELDING, Roy Thomas. ***Architectural Styles and the Design of Network-based Software Architectures**. 2000. University of California, Irvine*.

FILHO, Otávio Freitas Ferreira; FERREIRA, Maria Alice Grigas Varella. **Serviços semânticos: Uma abordagem RESTful**. Disponível em: < http://www.teses.usp.br/.../Dissertacao_Otavio_Freitas_Ferreira_Filho.pdf>. Acesso em: 30 ago. 2012.

HANSEN, Roseli; PINTO, Sergio Crespo. **Construindo ambientes de educação baseada na web através de *Web Services* educacionais**. Canoas, RS, 2003. Disponível em <http://www.nce.ufrj.br/sbie2003/publicacoes/paper07.pdf>. Acesso em: 03 ago. 2012.

O"Reilly, Tim. **REST vs. SOAP at Amazon**. Disponível em: <http://www.oreillynet.com/pub/wlg/3005 >. Acesso em: 3 out. 2012.

PRESCOD, Paul. ***Roots of the* REST/SOAP**. Disponível em: <http://www.prescod.net/rest/rest_vs_soap_overview/>. Acesso em: 30 ago. 2012.

RECKZIEGEL, MAURICIO. **Entendendo os *Web Services***. São Paulo, 2006. Disponível em: <http://www.imasters.com.br/artigo/4245/webservices/entendendo_os_webservices>. Acesso em: 25 abr. 2012.

SAMPAIO, CLEUTON. **SOA e *Web Services* em Java**. Rio de Janeiro: Brasport, 2006.

SAMPAIO, CLEUTON. **Web 2.0 e mashups: reinventando a Internet**. Rio de Janeiro: Brasport, 2007.

SANDOVAL, JOSÉ. **RESTFul Java *Web Services***. *Birminghan: Packt*, 2009.

SANTOS, Wagner Roberto dos. ***Web Services* WS-* vs. REST**. Disponível em:<http://netfeijao.blogspot.com/2009/05/web-services-ws-vs-REST.html/>. Acesso em: 21 ago. 2012.

SEELY, Scoot. **SOAP: cross platform *Web Services* development using XML**. New Jersey: Prentice Hall, 2002.

Souza, Ismayle de, Alcântara dos Santos, Pedro. **Automação de Testes de Desempenho e Estresse com o JMeter**. Disponível em: <http://www.ufpi.br/subsiteFiles/pasn/arquivos/files/artigoJMeter(1).pdf>. Acesso em: 14 out. 2012.

Stantchev, Vladimir. ***Performance Evaluation of Cloud Computing Offerings***. Terceira Conferência Internacional sobre Engenharia de Computação Avançada e Aplicações em Ciências: Berlin, 2009.

SUDA, Brian. **SOAP *Web Services***. 2003. University of Edinburgh.

TYAGI, Sammeer. **RESTful *Web Services***. 2006. Disponível em: <http://www.oracle.com/technetwork/articles/javase/index-137171.html>. Acesso em: 29 ago. 2012.

W3C. ***About the World Wide Web Consortium (W3C*)**. 2007. Disponível em: < http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626/>. Acesso em: 13 maio 2012.

Web Service. **In: WIKIPÉDIA, a enciclopédia livre**. [S.l.]: Wikipédia Foundation, 2006. Disponível em: <http://pt.wikipedia.org.br/wiki/Web_service>. Acesso em: 22 abr. 2012.