Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

ADRIEL FRANCISCO SANTIAGO CAVALEIRO

ESTILO ARQUITETURAL REST PARA CRIAÇÃO DE WEB SERVICES RESTFUL

ASSIS
2013

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

ADRIEL FRANCISCO SANTIAGO CAVALEIRO

ESTILO ARQUITETURAL REST PARA CRIAÇÃO DE WEB SERVICES RESTFUL

Trabalho de conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso Superior de Ciência da Computação

Orientador: Me. Douglas Sanches da Cunha

Área de Concentração: Informática.

ASSIS
2013

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

FICHA CATALOGRÁFICA

CAVALEIRO, Adriel Francisco Santiago

Estilo Arquitetural REST Para Criação De Web Services RESTFful/ Adriel Francisco Santiago Cavaleiro. Fundação Educacional do Município de Assis -- Assis, 2013

68p.

Orientador. Me Douglas Sanches Cunha.

Trabalho de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis.

1.Web Services 2.REST 3.JAVA 4.XML

CDD: 001.6

Biblioteca da FEMA

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

ESTILO ARQUITETURAL REST PARA CRIAÇÃO DE WEB SERVICES RESTFUL

ADRIEL FRANCISCO SANTIAGO CAVALEIRO

Trabalho de conclusão de curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisição do Curso de Graduação, analisado pela seguinte comissão examinadora:

Orientador: Me. Douglas Sanches da Cunha.

Analisador(1): Dr. Luiz Ricardo Begosso

Assis

2013

DEDICATÓRIA

Dedico este trabalho a Deus e aos meus pais e irmão.

AGRADECIMENTOS

Á Deus que me permitiu chegar até o fim me concedendo conhecimento, entendimento, condições, força e ânimo.

Aos meus pais por sempre me garantir boas condições, educação e pela grande paciência que tiveram durante esta jornada.

Ao Professor Douglas Sanches da Cunha pela orientação, por todo o conhecimento transmitido, não somente como orientador e pelo constante estímulo transmitido durante o trabalho.

À todos os meus amigos e colegas que de alguma forma ajudaram ou incentivaram durante todo este trabalho.

RESUMO

Este trabalho de pesquisa tem por finalidade descrever o estilo arquitetural REST para criação de web Services, através da definição das principais tecnologias e da arquitetura de Web Services. A partir do estudo da arquitetura geral dos Web Services é possível definir o modelo Arquitetural Orientado a Recursos(ROA), que difere dos modelos de web Services implementados em SOA (Service Oriented Architecture). As principais tecnologias presentes na Internet como a linguagem XML, para definição e estruturação de documentos e o protocolo HTTP, para envio de mensagens e definição de métodos de acesso e modificação à recursos torna possível a implementação de Web Services que possam ser mais próximos e parecidos com a Internet definidos como RESTful.

Palavras-Chaves: REST, ROA, HTTP, Web Services, RESTful

ABSTRACT

This work is indeed to describe the REST architectural style for Web Services impelentation, through the definition of the main Web Services Technologies. Starting by the study of the Web Services' general architecture it is possible to define the Resourse Oriented Architecture model (ROA), which differs from other Web Services Models like SOA(Service Oriented Architecture). The main Internet technoligies such as XML, for definition and data structure, and the HTTP protocol, for mensage exchanges and definitions of methods for resource access, it is possible to implemente Web Services look more like the actual Web, known as RESTful.

Key-Words: REST, ROA, HTTP, Web Services, RESTful.

Sumário

LISTA DE ILUSTRAÇÕES

1 – **INTRODUÇÃO**

Segundo Burke (2010), a Internet tornou-se essencial para o dia a dia de muitas as pessoas. Para ler jornais, pagar contas, fazer transações bancárias, comprar produtos e serviços, entre outras. Tudo isso através de uma aplicação comum como um browser, por exemplo. Até mesmo para conversar e socializar com outras pessoas em redes sociais como Facebook, Twitter, Linked-in, entre outras. Programadores e desenvolvedores, sempre procuraram maneiras de construir softwares e sistemas de forma distribuída, que seja reutilizável, escalável e livre para qualquer plataforma.

Seguindo esse princípio surgem os *Web services*, sistemas de software que podem estar implementados em um servidor qualquer e que podem ser acessados através da internet ou por meio da Intranet, por empresas de grande médio e pequeno porte. Pelo fato de um servidor atender a muitos acessos de forma rápida, muitos programas utilizam seus serviços oferecidos, como autorização de cartões de crédito, cálculo de frete para sistemas de compras, relatórios de trafego de veículos em um determinado local, ao invés de ter suas implementações (GURUGE 2004).

Os *Web services* implementam uma lógica para a aplicação que é definida por padrões de implementação, tais padrões podem ser CORBA, REST, SOAP entre outros. As aplicações são flexíveis, estendidas, modificadas e acima de tudo, possuem grande desempenho. Graças à *Web services* é possível hoje criar aplicações com certa rapidez e que podem ser utilizadas por dispositivos de várias plataformas (NEWCOMER 2002).

Web services não depende de plataformas específicas, isto é, qualquer *Web services* pode ser implementado em qualquer sistema operacional e ter seus serviços consumidos, utilizados por qualquer aplicação, não importando qual a linguagem de programação é utilizada, ou se a aplicação é desktop ou web, além de poderem ser comunicar com outros para obter seus serviços específicos (CERAMI 2002).

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

Para consumir, receber dados de entrada (input) e enviar informações (output) à *Web services*, utiliza-se de protocolos para representar dados, com o XML são especificados em um protocolo, como SOAP por exemplo. O SOAP é padronizado para compartilhamento de mensagens entre aplicações, sua especificação provê uma forma padrão de estruturar mensagens para um documento XML, e como esse pode ser implementado por qualquer linguagem de programação de qualquer sistema operacional, a troca de mensagens entre as aplicações e o serviço torna-se possível (SNELL, 2001).

Segundo Richardson (2007) apesar das grandes vantagens oferecidas por *Web services* mais tradicionais suas complexas operações e falta de suporte para diferentes formatos de mensagens tornaram sua programação um tanto quanto complicada. Graças ao REST, um renascimento do HTTP, surge como uma alternativa viável para evitar a complexidade imposta pelos modelos mais antigos como um simples conjunto de princípios que permite a desenvolvedores menos experientes conectar aplicações através de um estilo nativo da própria web.

Muitas aplicações necessitam de troca de informações, *Web services* tornam essas implementações possíveis, pois um dos seus princípios é gerar comunicação e compartilhamento entre hardwares, sistemas operacionais e linguagens de tipos diferentes. Tudo isso é possível devido à maneira com que eles obtêm e transmitem suas informações, eles trabalham na camada web, possuem uma linguagem própria para troca de informação entre si, utilizam XML para envio e retorno de mensagens, e ainda possuem um padrão para empacotamento delas.

1.1. Objetivo

Como principal objetivo, este trabalho irá apresentar o estilo arquitetural REST para criação de *Web services* e seus padrões de implementação, que permitem ampla utilização entre linguagens e fornece disponibilidade de serviços, os protocolos utilizados pelo modelo arquitetural e seu comportamento na Web. Após revisão bibliográfica da pesquisa, será desenvolvida uma aplicação que utilizará as tecnologias apresentadas, a fim de demonstrar algumas das funcionalidades citadas.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

1.2. Justificativa

O estudo das tecnologias que envolvem *Web services* e o estilo arquitetural REST se justificam pelo cenário atual da Internet que tem proporcionado a criação de aplicações distribuídas e de fácil implementação pelos demais desenvolvedores de software. A *Web services* possui um papel muito importante nesse cenário e, graças ao REST, foi possível criar novos modelos de serviços possuindo novos tipos de hipermídia para transferência de mensagens. Além de REST ser um estilo arquitetural relativamente novo que cresce em escala significativa se comparada ao avanço da internet, e as formas de processamento nas nuvens.

1.3. Motivação

Web services tornaram-se tão comuns que seus recursos são utilizados a todo momento sem nem mesmo serem percebidos, desde uma compra web, até mesmo um cadastro de clientes em um sistema de gerenciamento empresarial, onde por exemplo é feita uma consulta que retorna informações de endereço pelo CEP, em sistemas de calculo de frete, e outros.

Além de diminuir o tempo de desenvolvimento de um software, por utilizar serviços, podem garantir a integridade das informações que serão sempre atualizadas, sem necessidade de intervenção humana no software que o consome.

O fato de estarem ligados diretamente à interoperabilidade e alta disponibilidade motiva seu estudo, pois quanto mais a Internet cresce mais aplicações necessitam de integração de suas informações disponíveis para seu próprio aperfeiçoamento.

1.4. Perspectivas de Contribuição

Esta monografia pretende servir de referência a todos que queiram entender os conceitos e praticas de uso de um Web services, bem como sua implementação para serviços RESTful, ou seja, que utilizam por completo a tecnologia REST, pois

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

atualmente no mercado a maioria dos *Web services* são implementados em outras tecnologias como SOAP e WSDL e com isso será apresentado uma alternativa de serviço que será aplicada em Java.

1.5. Metodologia de Pesquisa

Para posterior definição, serão utilizados livros que descrevem a arquitetura de *Web services*, o estilo arquitetural REST e as demais tecnologias utilizadas para favorecer a criação de um serviço RESTful (que implementa todos os padrões de REST). Com base no conhecimento oferecido por tais livros, será elaborada uma pesquisa para definir *Web services*, como funcionam e como devem ser implementados, os padrões e tecnologias que definem REST, como HTTP, URI JSON e XML e em seguida será implementado um *Web services* em Java com a API JAX-RS, para desenvolvimento de *Web services* RESTful, de acordo com os conceitos definidos no estilo arquitetural *Representational State Transfer* (REST).

Este trabalho será organizado no formato de capítulos. O capítulo 2 apresentará as principais características dos *Web services*, as arquiteturas que os definem e algumas das tecnologias utilizadas. No próximo capítulo – capítulo 3, será apresentado o XML, sua estrutura, arquitetura e sua utilização no contexto dos *Web services*. O capítulo 4 apresentará o estilo arquitetural REST, sua arquitetura, funcionamento e métodos e sua relação com o protocolo HTTP para transferência de mensagens. No final do trabalho, no capítulo 5, será apresentada a criação de um *Web services* que implementa algumas funcionalidades do REST, e demonstra alguns dos aspectos abordados neste trabalho. Neste capítulo também serão apresentados aplicações clientes que farão a comunicação com o serviço.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

2 – WEB SERVICES

Existem algumas definições diferentes para determinar um *Web services*, alguns autores o definem como uma aplicação enquanto outros o definem como um padrão ou interface.

Segundo Booth et al.(2004), *Web services* é um sistema de software designado para suportar iterações interoperáveis de máquina-para-máquina sobre uma rede de computadores. Possui uma interface descrita em formato processável por máquinas (especificamente WSDL). Outros sistemas interagem com os *Web services* numa maneira pré-estabelecida por sua descrição usando mensagens SOAP, tipicamente transmitidas usando HTTP com uma serialização XML em conjunto com outros padrões da Web.

Este capítulo apresentará um panorama geral sobre *Web services*, sua arquitetura e seus modelos, assim como algumas das tecnologias utilizadas para sua implementação, dando um ensejo daquelas que serão utilizadas neste trabalho.

2.1 Definições

Basicamente, um "*Web services*" pode ser definido como uma interface acessível por redes de computadores (Intranet ou Internet) para funcionalidades de aplicações, construído usando os padrões da tecnologia da Internet (SNELL, 2001).

Segundo Newcomer (2002), *Web services* são aplicações XML mapeadas para programas, objetos ou funções que podem funcionar em desktop e serem usadas para integrações de regras de negócios. Eles utilizam esse formato de dados para criar documentos em forma de mensagens para enviar requisições através da rede e assim, opcionalmente, receber um documento contendo a resposta desejada. Suas especificações definem o formato de suas mensagens e como eles devem ser publicados e descobertos em rede. De uma maneira geral, *Web services* definem uma ligação padrão para conectar diversas funcionalidades de software.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

A Figura1, a seguir, ilustra a descrição de Newcomer de um *Web services* que permite o acesso a códigos de aplicações através de padrões da Internet.

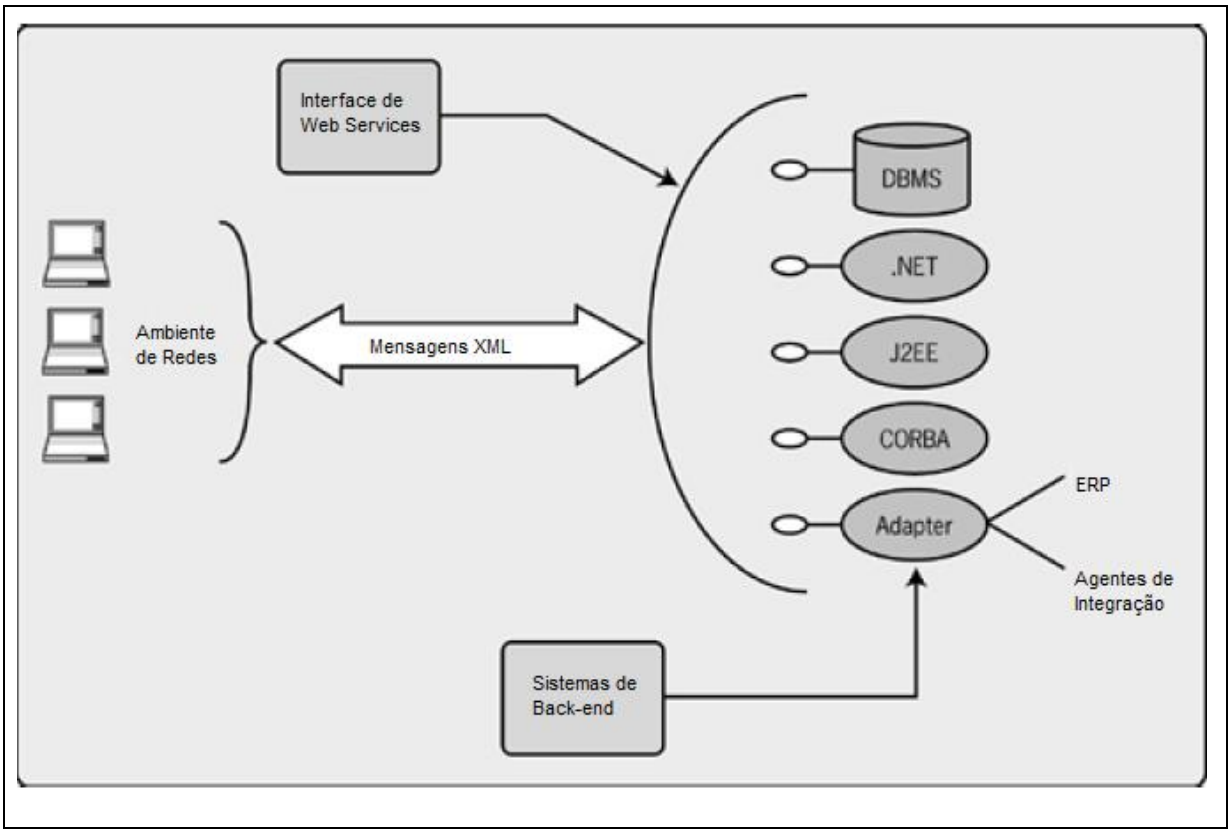

Figura 1 - Exemplo de acesso de *Web services*. (In: NEWCOMER, 2001, p.6).

Web services podem ter "aplicações" ou lógicas de aplicação modular, e independentes desenvolvidas por uma série de padrões. Tudo isso é imutável e indiscutível. São, normalmente, mapeados para programas, objetos, banco de dados ou funções abrangentes de regras de negócios. Eles fornecem a ligação padrão para conectar as funcionalidades de software (GURUGE, 2004).

Segundo Cerami (2002), um *Web services* é, portanto, qualquer serviço que está disponível em rede (Intranet ou Internet), que deriva XML para um sistema padrão de mensagens. Um *Web services* não depende de Sistema Operacional ou linguagem de programação e pode ser descoberto por mecanismos simples de busca.

Entretanto, um *Web services*, não é feito para ser uma aplicação completa, rica em recursos em seu próprio direito. Afinal, o intuito da utilização de *Web services* é tornar a programação distribuída e acessível para qualquer programa que necessite

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

dos serviços para se complementarem ou para a realização de determinada tarefa que necessite de cuidados mais específicos – apesar de não haver restrições sobre quão grande ou complexo ele deve ser (GURUGE, 2004).

Segundo Snell (2001) é importante ressaltar que *Web services* não necessita estar em um servidor específico para funcionar, pois, podem fazer parte de uma camada de mensagens na rede, seus únicos requerimentos são receber e enviar mensagens, usando alguma combinação de protocolos padrões da Internet. Por isso, *Web services* podem ser implementados por qualquer dispositivo em que os padrões da tecnologia da Internet possam ser utilizados.

2.2. Arquitetura

A arquitetura de *Web services* é definida por relacionamentos, conceitos e modelos, que são centrais para a interoperabilidade dos *Web services*.

Segundo Cerami (2002) uma maneira de visualizar a arquitetura dos *Web services* se dá por examinar o conjunto de protocolos que os definem.

Booth et at. (2004) afirma que *Web services* é um conceito abstrato que necessita ser implementado por um agente concreto. O agente é uma parte de software ou hardware, que envia e recebe mensagens. Enquanto o serviço é a fonte ou recurso que será caracterizada pela série de funcionalidades abstratas que ele provê. Essa abstração o torna imutável, pois vários agentes diferentes, ou linguagens de programação distintas, que podem acessá-lo sem que esse serviço mude.

Segundo Booth et al. (2004) o propósito de Web services é prover funcionalidades em favor de seu proprietário. Os serviços definem a entidade dos provedores, porque fornece um determinado agente para poder ser implementado.

O agente de requisição é o software ou hardware que deseja fazer uso das funções oferecidas pela entidade provedora de um Web service. São esses agentes que iniciam a troca de mensagens dos serviços. E para que essa troca de mensagens tenha sucesso é necessário que tanto as entidades de requisição e do provedor concordem com as semânticas e mecânicas estabelecidas pela descrição dos Web services (Booth et al., 2004).

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

A figura 2, ilustra o principal meio de interação com Web services. O cliente ou requerido serviço procura por serviços disponíveis e depois os invoca.

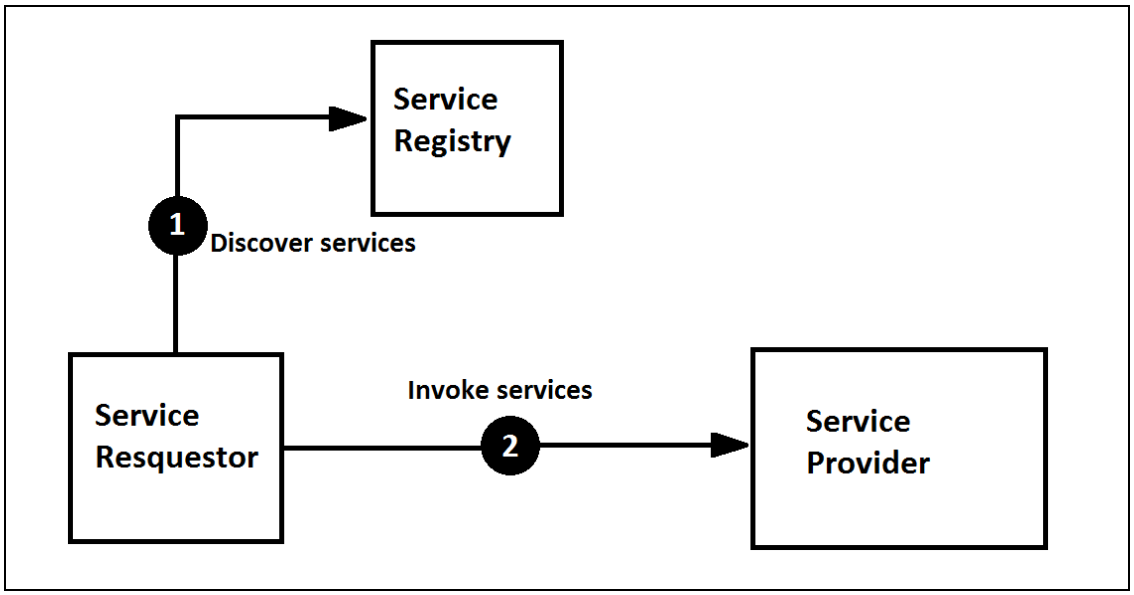

Figura 2 - Interações com *Web services* (In: SNELL, 2001, p.7) .

Para definir os protocolos da arquitetura dos *Web services* é necessário identificar as camadas que definem o conjunto arquitetural.

Segundo Cerami (2001), existem quatro camadas principais que definem a arquitetura dos Web services. Essas camadas são: Transporte de serviços – responsável por transportar mensagens entre as aplicações utilizando protocolos como HTTP, SMT ou FTP; Mensagens XML - responsáveis por codificar mensagens em arquivos comuns de XML e assim torna-las interpretáveis por ambos agentes. É nessa camada que se encontra o SOAP (*Simple Object Access Protocol*); Descrição dos serviços – é a camada responsável por descrever a interface pública de um *Web services* específico. Nessa camada são definidos os documentos WSDL (*Web services Description Language*); Descoberta de serviços – é a camada responsável por centralizar os serviços em um registro comum e assim facilitar os mecanismos de publicação e descoberta desses serviços.

Os mecanismos necessários para a troca de mensagens dos *Web services* são documentados em um *Web services Description* (WSD) que é uma especificação da interface dos *Web services* escrita em WSDL (*Web services Description Language*). Pode-se dizer que nele estão representadas as regras que governam

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

os mecanismos de interação entre os *Web services* (Booth et al., 2004).

2.2.1. Modelos Arquiteturais

Segundo Booth et al. (2004) existem atualmente quatro diferentes modelos arquiteturais dentro da arquitetura geral dos *Web services*. Em geral esses modelos compartilham alguns conceitos, porém o papel principal de cada modelo arquitetural é encapsular e expandir um aspecto particular da arquitetura geral.

O modelo *Message-Oriented* (Orientado a mensagens) tem o foco nas mensagens. A essência desse modelo fica em torno dos conceitos de agentes que enviam e recebem mensagens, a estrutura da mensagem em termos de cabeçalhos, corpo da mensagem, envelopes e os mecanismos para entregar a mensagem (Booth et al., 2004).

É nesse modelo em que são implementados os serviços baseados em SOAP. Segundo Richardson (2007) todos os Web services são orientados a mensagens, porque HTTP é orientado a mensagens e uma requisição HTTP é uma mensagem com envelope contendo um documento em seu interior. Serviços baseados em SOAP exigem de seus clientes um segundo envelope (um documento SOAP) dentro do envelope HTTP.

A Figura 3 na próxima página, descreve o modelo arquitetural orientado a mensagens, a interação entre os clientes e a mensagem e os componentes das mensagens.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

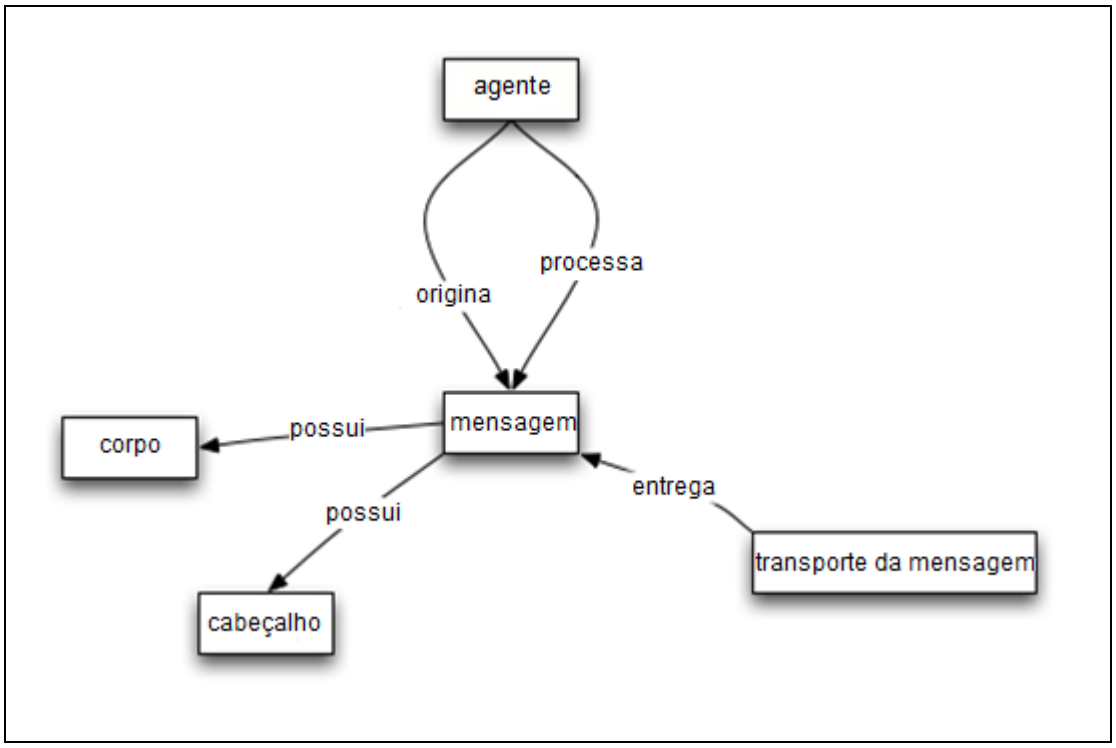

Figura 3 - Modelo Orientado à Mensagens(In: Booth et al., 2004).

O segundo modelo a ser observado é o *Service-Oriented* (orientado a serviços) que é focado nos aspectos de serviço e suas ações. O modelo orientado a serviços é o mais complexo de todos os modelos arquiteturais. Entretanto, baseia-se apenas algumas ideias principais. Um serviço será liberado por um agente e usado por outro. Serviços são mediados pelos motivos das mensagens trocadas pelos agentes de requisição e de publicação (Booth et Al., 2004).

Esse modelo arquitetural utiliza meta dados que servem para documentar vários aspectos dos serviços: detalhes da interface e ligações de transporte às semânticas do serviço e quais regras e restrições podem existir no serviço (Booth et Al., 2004).

Segundo Richardson (2007) a melhor definição para SOA (*Service Oriented Arquitecture*) seria de uma arquitetura de software baseada na produção e no consumo de Web services e deveria ser descrita como um modelo que usa serviços como componentes de um software.

A Figura 4 apresenta o modelo arquitetural Orientado a serviços (SOA), o papel dos agentes, mensagens e meta-dados sobre o serviço.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

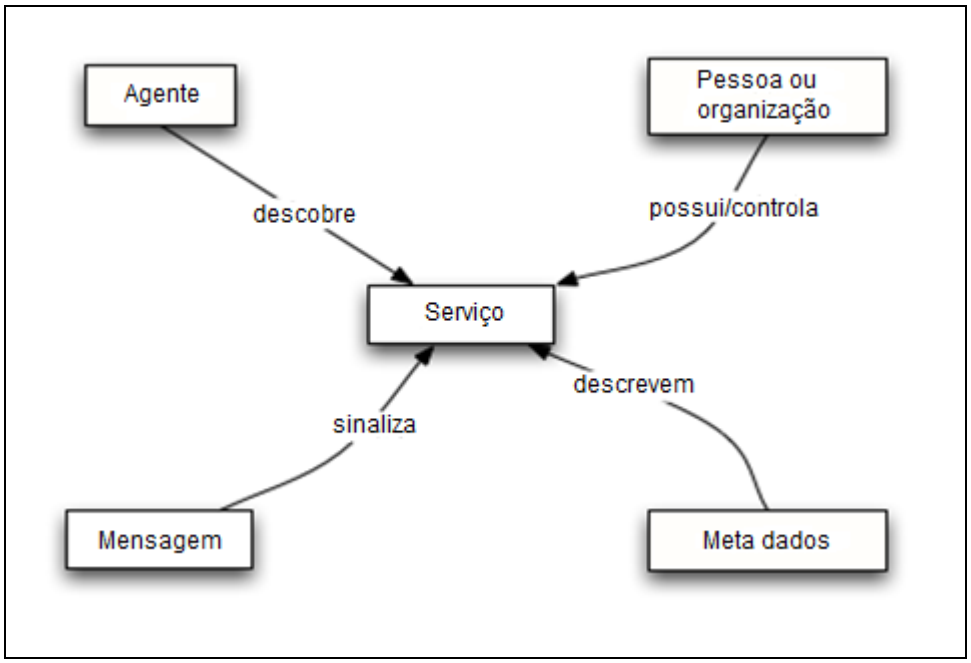

Figura 4 - Modelo Orientado a Serviços (In: Booth et al., 2004).

O próximo modelo que será observado é o *Resource-oriented* (ROA) ou orientado a recursos, que utiliza os conceitos da arquitetura da Web para sua implementação. O foco desse modelo é relacionado aos recursos – documentos ou serviços que podem ser acessados através de URIs, e suas representações – objetos de dados que refletem o estado de um recurso (Booth et al., 2004).

Web services RESTful são implementados por esse modelo arquitetural. ROA e RESTful serão apresentados com mais detalhes no capítulo 4.

A figura 5 apresenta, de modo simplificado, os principais agentes presentes no modelo ROA: o recurso, sua URI (identificador), sua representação e a pessoa ou organização que o publicou.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

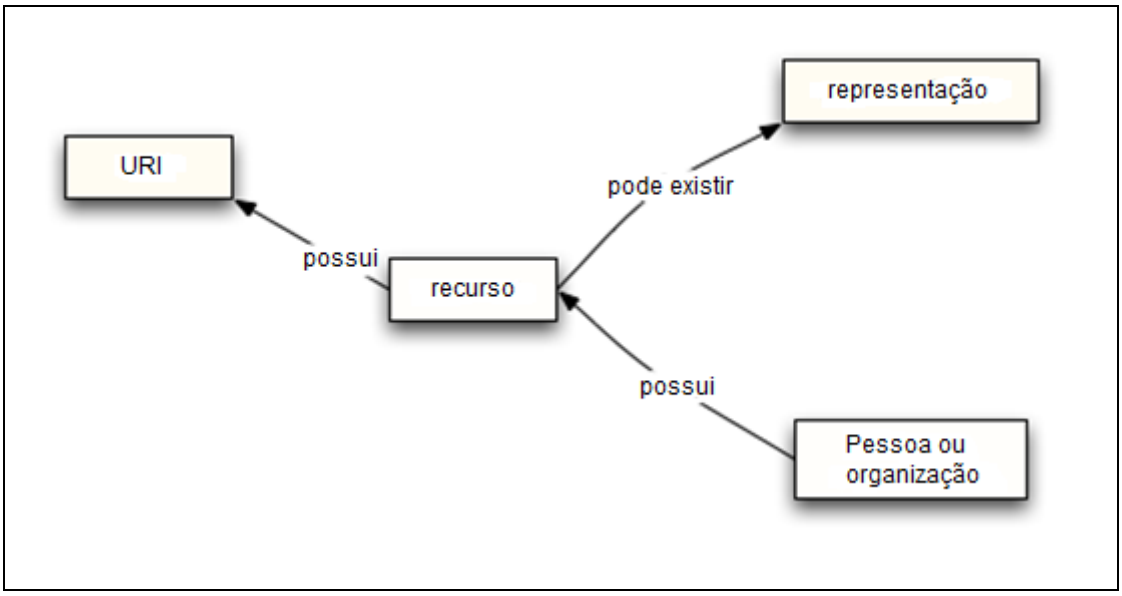

Figura 5 - Modelo Orientado a Recursos(In: Booth et. al, 2004).

O último modelo arquitetural é o Modelo de Políticas (*Policy Model*), que tem como principal objetivo as restrições dos comportamentos dos agentes e dos serviços. É normalmente generalizado com recursos, pois as regras podem ser aplicadas em documentos ou em recursos computacionais ativos (Booth et at., 2004).

Segundo Booth et al. (2004), esse modelo apresenta as preocupações de segurança, qualidade dos serviços, gerenciamento e aplicação. Apesar das regras serem aplicadas aos recursos, elas também pode ser aplicados nos agentes que tentam acessar tais recursos.

A figura 6 a seguir ilustra a relação entre as regras os agentes, os recursos e as ações que podem ser executadas.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

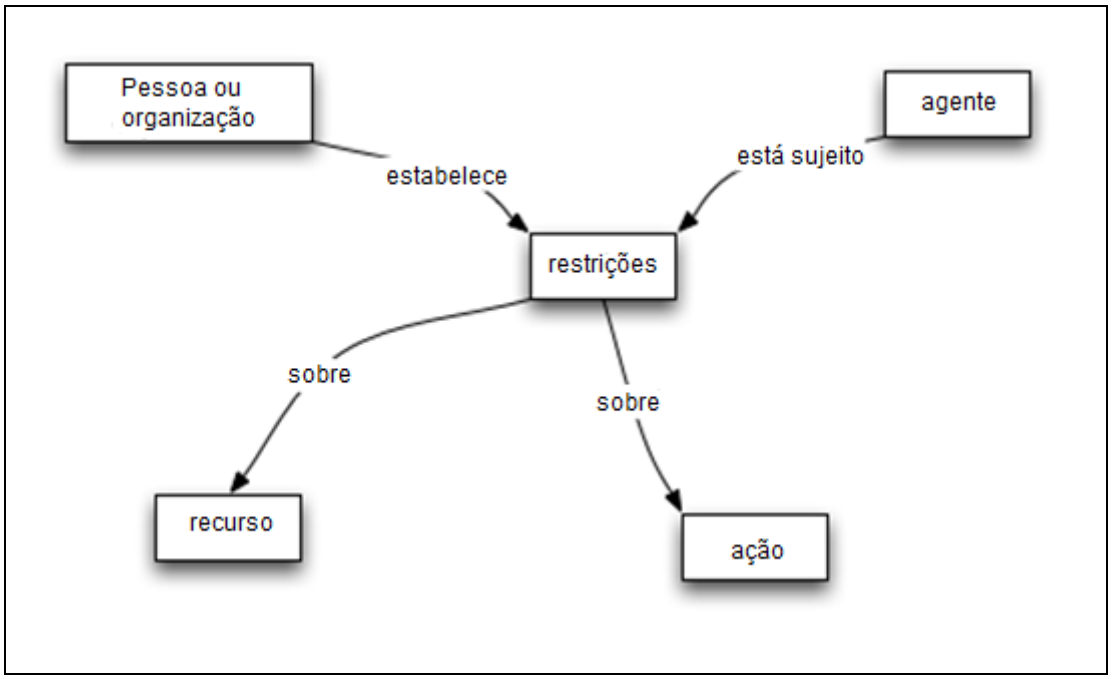

Figura 6 - Policy Model(In: Booth et al., 2004).

2.3. Tecnologias Dos Web services

A arquitetura dos *Web services* possui muitas camadas e tecnologias inter-relacionadas. Da mesma forma que existem vários modos para construir *Web services* e visualizar essas tecnologias (Booth et al., 2004).

Nesta sessão serão descritas algumas das tecnologias que são mais importantes na arquitetura geral.

A figura 7 apresenta algumas das tecnologias utilizadas na arquitetura geral e as camadas em que elas atuam.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

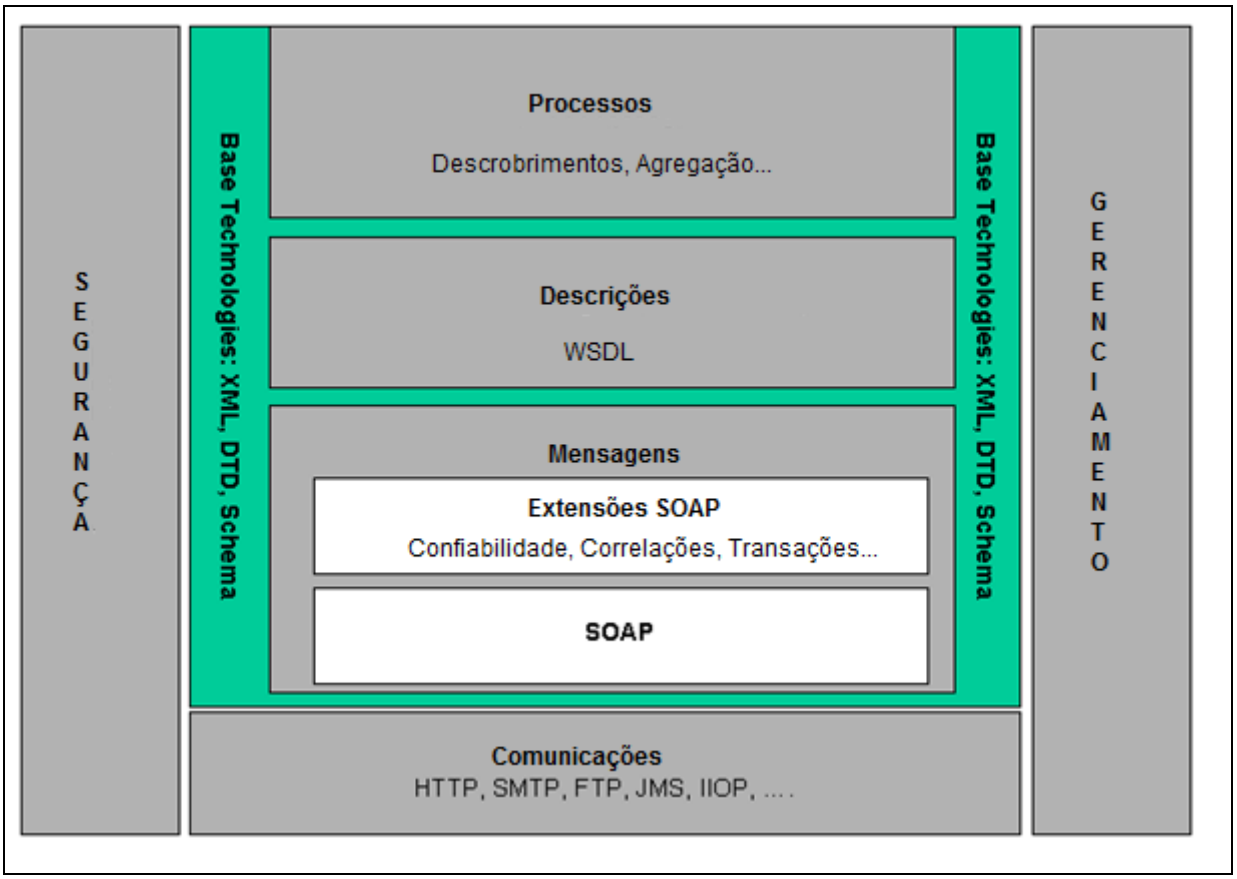

Figura 7 - Principais camadas da Pilha arquitetural dos Web services (In: Booth et al., 2004).

Na figura acima é possível identificar as principais camadas que atuam em aplicações que interagem com a Web. Na camada de segurança estão os roteadores e os firewalls que dão acesso à Internet. A camada de administração representa as aplicações de alto nível que fazem a comunicação com os usuários.

Na camada central da figura é onde ocorre a Web services. Essa camada se divide em duas partes; a primeira é designada para as tecnologias bases da Web services. Tais tecnologias bases incluem XML, SOAP, WSDL e os processos de descoberta, agregação e ordenação. A segunda parte dessa camada demonstra os protocolos utilizados para a transmissão de informações pela rede como o SMTP e o HTTP.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

2.3.1. HTTP

O HTTP (Hypertext Transfer Protocol) ou protocolo de transferência de hipertexto. É um protocolo em nível de aplicação para sistemas distribuídos, colaborativos e de informação de hipermídia. HTTP é usado pela Web desde 1990 e também é usado como protocolo de comunicação entre agentes de requisição e agentes provedores por sistemas da Internet (FIELDING et at., 1999).

O protocolo HTTP possui uma pequena e fixada série de operações e de interface simplificada que asseguram a comunicação entre clientes e serviços. Essa interface é definida por seis métodos principais (GET, PUT, DELETE, POST, HEAD e OPTIONS) (BURKE, 2010).

Segundo Burke (2010) qualquer requisição de Web services envolve os três mesmos passos: informar quais dados serão utilizados na requisição HTTP; Formatar os dados como uma requisição HTTP e enviar para o servidor HTTP apropriado; analisar os dados de resposta – código da resposta e cabeçalhos – de acordo com as estruturas de dados que seu sistema necessita.

O protocolo HTTP será analisado com mais detalhes no capítulo 4 e demonstrado no capítulo 5 juntamente com a aplicação para testes.

2.3.2. SOAP

Dentro do pacote de tecnologias dos Web services, SOAP atua como um protocolo padrão para envelopar as mensagens compartilhadas pelas aplicações. Sua especificação o define como um simples envelope baseado em XML para informações que serão transferidas e uma série de regras para tradução desse documento para XML (SNELL, 2001).

A figura 8 descreve a estrutura e o formato de um envelope SOAP que consiste de duas partes principais: o cabeçalho do envelope, que contém informações como dos agentes e do tipo de método que será utilizado no serviço, e o corpo do envelope que contém a mensagem.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

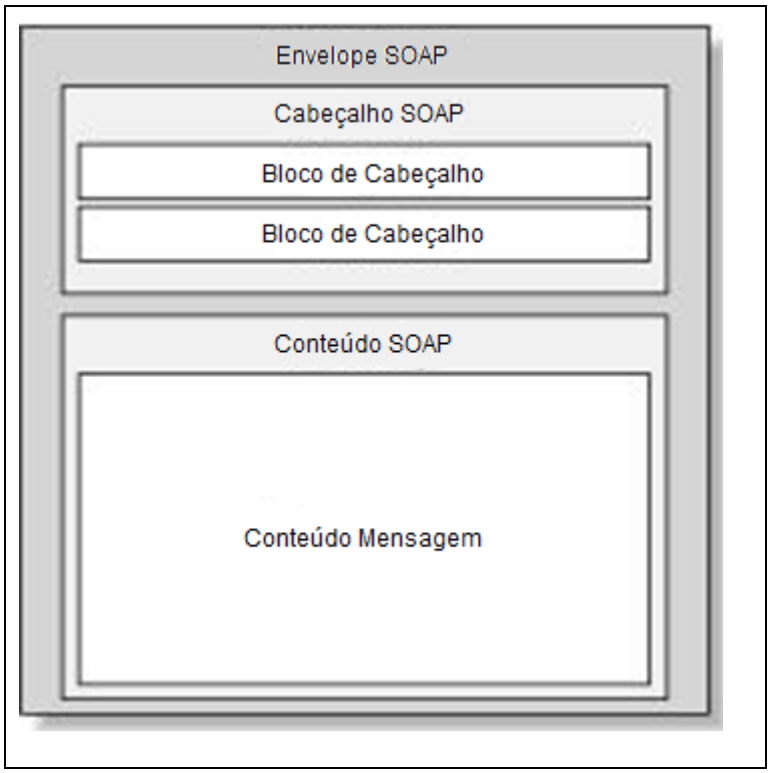

Figura 8 - Envelope de Mensagem SOAP(In: SNELL, 2001 p.17).

2.3.3. WSDL

Web services Description Language (WSDL), ou Linguagem de Descrição de Web services é um modelo de XML para descrever Web services. Segundo Newcomer (2002), WSDL é o coração de Web services baseados em SOAP, fornecendo uma maneira comum para representar os tipos de dados passados pelas mensagens, as operações que serão realizadas nas mensagens e o mapeamento das mensagens nos meios de transporte da rede. Nos documentos WSDL é também especificado como os Web services são estruturados: as definições de tipos de dados, as operações abstratas e as ligações dos serviços.

2.3.4. REST

Representational State Transfer (REST) é um estilo arquitetural que especifica regras, como de uma interface uniforme, que, se aplicadas a *Web services*, induzem propriedades desejáveis, como desempenho e escalabilidade permitindo

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

que os serviços funcionem de forma mais eficiente na Web (ORACLE, 2013).

Segundo Burke (2010) REST e SOAP são diferentes. SOAP é um protocolo de mensagens, enquanto REST é um estilo arquitetural de software para sistemas distribuídos de hipermídia. Ou seja, são sistemas nos quais texto, gráficos, áudio e outras mídias armazenadas através da rede e interconectados por hiperlinks. A Internet é um claro exemplo de tal sistema.

Conforme dito anteriormente, serviços em REST implementam o modelo arquitetural orientado a recursos. REST será abordado com mais detalhes no capítulo 4.

Este capítulo abordou algumas das principais tecnologias, arquiteturas e o funcionamento geral dos *Web services*. Das tecnologias abordadas, HTTP e XML serão demonstradas com mais detalhes neste trabalho além do modelo arquitetural Orientado a Recursos.

O próximo capítulo apresentará uma visão geral sobre XML, sua arquitetura, funcionamento e utilização o desenvolvimento de Web services.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

3 – XML

Segundo Guruge(2004), XML é a espinha dorsal dos *Web services* por descrever suas tecnologias e por ser uma das tecnologias mais utilizadas e presentes na Internet.

Este capítulo irá descrever a arquitetura do XML, seu funcionamento e importância nos Web services.

3.1. Visão Geral

XML (*Extensible Markup Language*) é uma linguagem de marcação criada para gerar documentos que podem servir para dois tipos principais. O primeiro, como um arquivo de representação de baixo nível para descrever, por exemplo, arquivos de configuração. O segundo, como um meio de adicionar meta-dados a documentos, similar ao HMTL (FAWCETT, 2012).

Segundo Newcomer (2002), XML é similar ao HTML possuindo elementos, atributos e valores. Porém, HTML possui um número finito de elementos e atributos, enquanto XML permite a definição de qualquer quantidade de elementos e atributos. Documentos XML bem estruturados podem ser facilmente exibidos em navegadores Web.

A característica do XML de definir atributos e elementos é derivada como um subconjunto de outra definição, mais antiga, conhecida por SGML (*Standard Generalized Markup Language*), que é considerado o primeiro passo para o surgimento de outras tecnologias de linguagens de marcação mais eficientes. SGML é o primeiro avanço da definição de criar marcações efetivas. Ao invés de possuir uma série de marcações pré-definidas e que não podem ser alteradas ele transmite como as marcações devem ser definidas, permitindo que usuários criem suas próprias marcações. Assim é possível defini-las usando uma aplicação de sintaxe padrão em que qualquer analisador SGML possa consumir os documentos e interagir com eles (FAWCETT, 2012).

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

Segundo Fawcett (2012), apesar de todas as vantagens de SGML e sua capacidade de criar vários tipos de marcação, ele era um pouco complicado e não existiam muitas aplicações que pudessem ler suas definições de marcação e usá-las para processar documentos corretamente. XML foi desenvolvido para manter a flexibilidade do SGML sem a maior parte de sua complexidade, pois XML não dita o formato geral de um arquivo ou quais meta-dados podem ser adicionados. XML simplesmente especifica algumas poucas regras.

Segundo Guruge (2004), XML pode ser usado com qualquer tipo de dados, independentemente da natureza dos dados. XML fornece um framework para descrever dados e assim tornar os dados auto descritivos.

2.2. Arquitetura

XML manipula dados no formato de documentos. Segundo Guruge (2004), toda transferência de dados entre *Web services* deve ser necessariamente no formato de um documento XML.

Os dados são incluídos nos documentos em forma de *Strings* de texto. Segundo Bray (2004), cada documento XML possui uma estrutura lógica e física. Faz-se necessário ressaltar que essas estruturas devem funcionar paralelamente. Fisicamente, os documentos são compostos de unidades chamadas entidades, ou unidades de armazenamento. Uma entidade pode referir-se a outras, incluindo-as no documento. Qualquer documento começa por uma entidade raiz. Logicamente, os documentos são compostos por elementos, comentários, declarações e instruções de processamento que estão indicados explicitamente no documento.

Os blocos de construção mais básicos do XML são os elementos. Elementos são unidades específicas de dados que, juntamente com as marcações, tem como função descrever dados e informações dentro dos documentos.

Os elementos em XML são delimitados por *tags* (etiquetas) que se resumem em *tag* de início, com os símbolos <> e *tags* de fim, com os símbolos</>. As *tags* são similares ao HTML e são definidas através de tais símbolos. É a *tag* de início que define o nome do elemento (GURUGE, 2004).

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

No exemplo <nome>Adriel</nome>, a denominação do elemento é "nome" e o seu conteúdo é "Adriel".

Apesar de XML garantir ao desenvolvedor a liberdade para criar elementos específicos, existem regras e restrições para a composição dos nomes dos elementos XML. Segundo Guruge (2004), os elementos só podem começar por letras (A a Z) ou por um sinal *underline* (_) e após o primeiro caractere podem existir outras letras, números, pontos (.), hífen (-) e dois pontos (:). Deve-se destacar que não podem existir espaços nos nomes dos elementos e as denominações não podem começar com "xml".

Uma diferença arquitetural do XML para o HTML consiste no fato de XML ser *case sensitive*, ou seja, letras maiúsculas e minúsculas diferem na análise dos elementos. Dessa forma, <nome>, <NOME> e <Nome> representam três elementos diferentes. Assim <Profissao></profissao> não é um elemento válido em XML (Guruge, 2004).

XML permite o uso de elementos dentro de outros elementos, formando assim o conceito de atributos.

A figura 9 demonstra o funcionamento de um elemento com seus atributos.

```
<pais>
 <nome-do-pais>Brasil</nome-do-pais>
<lingua-oficial>Portugues</lingua-oficial>
</pais>

<pais nome-do-pais:"Brasil" lingua-oficial:"Portugues"/>
```

Figura 9 - Elementos e atributos XML.

A última *tag* do exemplo define uma *tag* "vazia" Segundo Guruge (2004), a habilidade de possuir atributos permite ao XML a possibilidade de criar elementos "vazios", ou seja, elementos que possuem uma única *tag* como início e fim e que possui os atributos dentro de seu corpo. São *tags* mais simples e de fácil entendimento e são, principalmente, usadas nas definições de *namespaces* em XML.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

Apesar de XML restringir o uso de alguns caracteres especiais, existem casos em que é necessário utilizar alguns símbolos na definição do corpo do atributo. Com a utilização de entidades é possível definir os caracteres que serão utilizados no documento. Quando são analisadas pelo interpretador XML, as entidades são lidas de uma maneira diferente e são substituídas pelo seu valor que pode ser definido em códigos de UNICODE (GURUGE, 2004).

A Figura 10 demonstra a definição de uma nova entidade e a sua utilização no corpo do documento. O analisador substitui o comando "&copyright" pelo símbolo que é representado pelo seu código "&#233". Assim, quando o código for executado ele exibirá o símbolo "©" (GURUGE, 2004).

```
<!ENTITY eacute "&#233;">


<article-body>
&copyright; Anura Gurug&eacute;
</article-body>
```

Figura 10 – Exemplo de entidade XML(In: GURUGE 2004).

2.3. Funcionamento

Segundo Newcomer (2002), os elementos e atributos em XML definem os tipos e estruturam as informações para os dados que eles carregam, incluindo a capacidade de modelar dados e definir uma estrutura específica para um determinado domínio.

Por causa da grande flexibilidade proporcionada por XML, é comum existir a preocupação de um documento ser entendido por ambas as partes que interagem com um sistema.

A figura 11 apresenta um exemplo da flexibilidade do XML, em que uma informação pode ser descrita de várias maneiras diferentes.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

```
<phoneNumber>(123) 456-7890</phoneNumber>
<phoneNumber>
    <areaCode>123</areaCode>
    <exchange>456</exchange>
    <number>7890</number>
</phoneNumber>
<phoneNumber area="123"  exchange="456"  number="7890" />
<phone area="123">
    <exchange>456</exchange>
    <number>7890</number>
</phone>
```

Figura 11 - Representação de Informação em XML (In: SNELL, 2001, p.17).

No contexto de Web services, XML não é somente utilizado como um formato para descrever mensagens, mas também como uma maneira de descrever a forma com que os serviços foram definidos. Assim, é importante que os agentes envolvidos possam entender o significado dos mesmos elementos e da mesma forma (NEWCOMER, 2002).

O principal objetivo de XML é garantir que seus atores (interpretadores, *web browsers, Web services* e outras aplicações que utilizam XML) sejam capazes de, facilmente, entender, sem ambiguidade e com consistência, a natureza e estrutura dos dados dentro do documento, sem confundir o que os dados podem representar. Possuir um documento XML bem formado não garante que ele será interpretado por qualquer aplicação (GURUGE, 2004).

Segundo Guruge (2004), XML fornece mecanismos de DTD (Definição de Tipo de Documento) e XML *schemas* (esquemas) para facilitar e garantir entendimento mútuo para ambas as partes envolvidas numa transação dos documentos.

Assim como SGML, XML também possui o conceito de DTD. Ele define aquilo que o documento precisa apresentar e as estruturas permitidas dentro do documento XML. É também a especificação para qualquer marcação que será usada e todo o documento que possa aparecer deve ser declarado no DTD (GURUGE, 2004).

A figura 12 define a estrutura básica da declaração de um DTD. Os símbolos "?", "*" e "+" indicam, respectivamente, que o conteúdo da descrição pode ocorrer zero ou uma vez; zero ou mais vezes; e, uma ou mais vezes:

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

```
<!ELEMENT element_name (content_description) ['?' | '*' | '+']>
```

Figura 12 – Declaração de Uma Entidade DTD(In: GURUGE 2004).

Em uso real de aplicação o DTD pode ser definido conforme a figura 13.

```
<!ELEMENT person (name, company*)>
<!ELEMENT name (salutation?, first_name, middle_name*,
last_name)>
<!ELEMENT salutation (#PCDATA)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT middle_name (#PCDATA)>
<!ELEMENT last_name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
```

Figura 13 – Declaração Completa DTD(In: GURUGE 2004).

O valor "#PCDATA", tido em alguns dos elementos, define que tal elemento possuirá uma sequência de texto, ou seja, os elementos definidos com esses atributos não podem ter sub elementos (GURUGE, 2004).

Para garantir a consistência de um documento XML existem, além dos DTDs, os esquemas XML (*schemas*).

Um esquema XML pode conter uma descrição mais compreensiva e rigorosa do conteúdo de um documento tanto de maneira modular, quanto de maneira orientada a objetos. Definir um bom esquema XML é mais complexo do que criar um DTD, pois esquemas exigem que os dados sejam rigorosamente especificados (GURUGE 2004).

Segundo Vlist (2002), um esquema XML é uma formalização das restrições expressadas como regras, Ou então, um modelo de estruturas, que se aplica a uma classe de documentos XML. De várias formas os esquemas fornecem um framework no qual as implementações podem ser construídas. Formalizar as estruturas e regras tornam as aplicações cada vez mais diversificadas, porém, a maioria dos esquemas, pode ser classificada como de validação ou de documentação.

Os esquemas de validação são os mais comuns, já que a validação é, geralmente, o

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

propósito inicial e primário dos esquemas. Validar os documentos em esquemas garante que o conteúdo do documento apresentará as regras esperadas pela implementação, assim, o código necessário para processá-las será simplificado (VLIST, 2002).

Além de validação, os esquemas são frequentemente usados para documentar os vocabulários do documento, mesmo quando a validação não é requerida. Os esquemas fornecem uma descrição formal dos elementos presentes no documento que pode ser de difícil entendimento. É muito comum publicar a especificação de um novo elemento no esquema do documento. Por isso, a partir da complexidade das descrições formais de esquemas bem definidos, é possível gerar uma documentação de fácil entendimento que podem ser visualizados por ambientes de desenvolvimento integrado as IDEs (*Integrated Development Environment*) (VLIST, 2002).

2.4. Mensagens

Segundo Newcomer (2002), XML é o fundamento de criação dos Web services, pois fornece a descrição, o armazenamento e o formato de transição para a troca de mensagens, além de ser fundamental para as tecnologias que transferem os dados.

Conforme abordado no capítulo 2, nos Web services baseados em SOAP, XML é a tecnologia utilizada para descrever e definir as mensagens utilizadas pelas transações. É com XML que são definidos os cabeçalhos e o corpo da mensagem para assegurar a comunicação entre os agentes dos Web services.

A figura 14 demonstra como o XML está relacionado com o funcionamento dos Web services.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

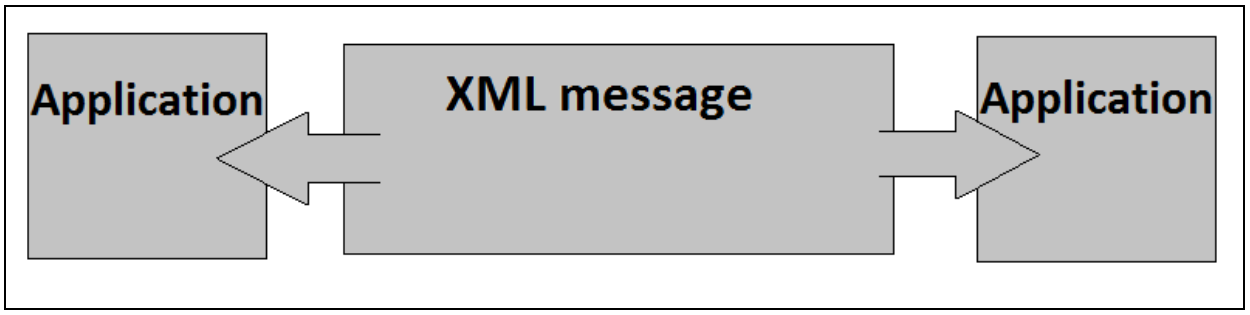

Figura 14 - As mensagens XML (In: SNELL, 2001).

Segundo Guruge (2004), a maioria dos Web services atuais são baseados ou implementam XML de uma forma ou de outra. Toda transferência de dados entre os agentes de um Web services, na maioria das vezes, deve ser feita no formato de um documento XML.

Nos Web services REST os serviços são manipulados pelas trocas de representações XML dos recursos. Para isso, utiliza-se uma série uniforme e restrita de operações, tornando, assim, o significado da mensagem não dependente do estado da conversação. Esse uso de representações garante a integridade dos dados e melhora o desempenho e a escalabilidade das transações (Booth et al., 2004).

XML é uma das mais importantes tecnologias utilizadas na Internet seu uso, conforme mencionado anteriormente, é imprescindível para a arquitetura dos *Web services*. Tanto para a definição das especificações que definem os serviços (WSDL) quanto para a estruturação das mensagens transmitidas pelos clientes e serviços.

No caso do REST, apesar de existirem outras formas de estruturar os dados, como é o caso do JSON, XML pode ser usado nas mensagens para estruturar as representações trocadas pelos serviços e clientes.

O próximo capítulo apresentará o estilo arquitetural REST, suas especificações, funcionamento e arquitetura derivada arquitetura de Web services orientada a recursos.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

4 – REST

Este capítulo apresentará o estilo arquitetural de implementação de Web services REST. Serão feitas a conceituação do termo e a apresentação da arquitetura dessa implementação.

4.1. Definição

REST (Representational State Transfer), ou transferência de estado representativo, é um estilo arquitetural para criação de aplicações distribuídas de hipermídia. São sistemas nos quais qualquer tipo de mídia armazenada na rede e interconectada por hiperlinks pode ser transferida entre os agentes. A própria Internet é o melhor exemplo desse sistema (KALIN, 2009).

De fato, o principal objetivo dos *Web services* RESTful (que implementam por completo o estilo arquitetural REST) é assegurar que as aplicações sejam mais próximas e parecidas com a WEB. Dessa forma, os sistemas computacionais podem ser mais escaláveis, de fácil modificação e com performance melhorada para transações (RICHARDSON, 2007).

Diferente dos serviços baseados em SOAP, os Web services RESTful utilizam o protocolo HTTP como parte principal e fundamental de sua arquitetura. Nos demais serviços apresentados ao longo deste trabalho, HTTP é utilizado como um simples protocolo com métodos para transferência de mensagens entre agentes, ou como uma maneira de transpassar firewall (KALIN, 2009).

Para que REST funcione corretamente é necessário à implementação dos quatro principais princípios arquiteturais que o definem. Tais princípios se resumem em: endereçar e identificar todos os recursos apresentados pelo serviço por URIs; utilizar representações para iterações entre os agentes; utilização de uma interface uniforme para manipulação dos recursos apresentados pelo sistema – os métodos HTTP fornecem tal interface; tornar as iterações sem estado - assim uma não depende de outra anterior para sua execução, assim a aplicação torna-se mais

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

escalável (Oracle, 2013).

Os princípios citados acima serão descritos com mais detalhes na próxima sessão.

4.2. Arquitetura

Conforme abordado no capítulo 2, REST implementa o modelo arquitetural orientado a recursos (ROA), que possui o foco nos aspectos da arquitetura que se relacionam com os recursos (BOOTH, 2004).

Pode-se afirmar que um recurso é qualquer elemento que possa ser referenciado como uma "coisa". Normalmente essa "coisa" é algo que deve ser representado por uma sequencia de bits, seja ele um objeto físico como uma bola, ou um conceito abstrato como uma emoção (RICHARDSON, 2007).

Para que os recursos possam ser acessados pelas entidades que interagem com os serviços é necessário que exista alguma identificação para eles. Em serviços RESTful, os recursos são identificados por URIs(*Uniform Resource Identifier*) que funcionam como o nome e o endereço dos recursos. Se uma informação não possui URI então ela não é um recurso e por isso não está na Web (RICHARDSON, 2007).

Segundo Fielding (2000), por definição, os recursos podem ter um ou mais URIs, contudo recursos distintos não podem possuir a mesma identificação. As URIs podem definir tanto recursos estáticos como recursos que variam. Com isso pode ocorrer de um recurso possuir mais de um identificador, por exemplo, em um programa na qual uma URI define a versão mais atual, enquanto outra URI define especificamente a versão 2.3.2 do sistema.

Segundo Richardson (2007), URIs também devem ser auto descritivas possuindo uma correspondência que seja intuitiva. Dessa maneira, em sua estrutura, as URIs variam de forma preditiva. Pois, conhecendo a estrutura principal do serviço, há maior facilidade em determinar quais as possíveis operações para cada recurso.

4.2.1. Endereçamento

A identificação de todos os recursos disponíveis no serviço é de extrema importância

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

para que o princípio de endereçamento dos Web services RESTful possa ser implementado.

Segundo Richardson (2007), uma aplicação pode conter um número infinito de URIs, pois todos os dados expostos como recursos precisam conter URIs para serem acessados. É esse endereçamento que possibilita à REST um menor consumo de tráfego de rede, pois um recurso endereçável pode ser salvo em cache para ser reutilizado em outro momento.

Segundo Fielding (2000), a vantagem de se utilizar os recursos de cache está em aumentar a eficiência da aplicação sobre uma rede, aumentando a escalabilidade e diminuindo a latência de uma série de interações. Recursos em cache podem reduzir, ou até mesmo eliminar, transições desnecessárias entre o cliente e o serviço. Porém, podem diminuir a integridade das informações, pois um recurso salvo em um cache pode estar desatualizado em relação à sua representação no serviço.

HTTP e a própria Internet são endereçáveis, tornando possível copiar hiperlinks para posterior utilização. Se HTTP não tivesse tal característica, seria necessário copiar todo o conteúdo de uma página Web para utilização em outro serviço. Dessa forma, seria impossível informar qual recurso de um Web services deve ser utilizado para determinada aplicação (RICHARDSON, 2007).

4.2.2. Interações sem Estado

A partir da definição do endereçamento dos recursos, surge o princípio de deixar todas as transações sem estado. Todas as requisições entre o cliente e o servidor devem conter todas as informações necessárias para completar a requisição. Isso pode reduzir o desempenho devido ao envio de dados repetitivos em uma série de requisições. Com base nisto, foram definidos os recursos de cache para a arquitetura (FIELDING, 2000).

Segundo Richardson (2007), todas as requisições HTTP acontecem em total isolamento, ou seja, mesmo que o cliente envie para o servidor todas as informações necessárias para a requisição, o serviço nunca usará informações de

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

requisições antigas para terminar a interação. Isso é comumente visto em sites em que o botão "voltar" dos browsers funcionam de uma maneira diferente exigindo que uma nova requisição ao serviço seja feita.

O princípio de manter as interações sem estado melhora a escalabilidade do serviço Web, pois cada interação dura somente uma requisição. Permitindo assim, que o serviço libere os recursos com mais rapidez, simplificando a implementação dos recursos. A integridade das informações também é melhorada pela simplificação da recuperação de falhas parciais (FIELDING, 2000).

4.2.3. Representações

O terceiro princípio da arquitetura e o principal mecanismo para descrever o estado de um recurso são as representações. Em serviço SOAP os clientes interagem diretamente com os dados contidos no serviço através das mensagens.

Representações são objetos de dados que refletem o estado atual de um recurso, podendo afirmar que um recurso pode ter mais de uma representação (BOOTH et al., 2004).

Segundo Richardson (2007), um recurso é apenas uma ideia ou uma interface de como os dados devem ser representados. Para isso as representações são definidas, para implementar a descrição dos recursos em uma linguagem ou formato específico.

Segundo Fielding (2000), representações servem para manter a integridade da mensagem. São definidas como uma sequencia de bytes que possui uma sequencia de meta-dados para descrever os *bytes*. O formato dos dados é conhecido como *media types*, ou tipos de mídia, e podem ser processadas automaticamente pelo receptor da mensagem ou renderizadas para a visualização de um usuário.

4.2.4. Interface Uniforme

O último princípio da arquitetura, e também o que diferencia REST dos demais

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

estilos de Web services, é o de uma interface uniforme entre os componentes (FIELDING, 2000).

Segundo Fielding (2000) generalizar a interface do sistema torna a estrutura geral simples e a visibilidade das interações é aumentada. Contudo, é possível que a eficiência diminua uma vez que toda a informação é transferida de uma forma padrão que talvez não seja a mais adequada para as necessidades da aplicação. Por isso, REST é designado a ser eficiente em uma transferência de hipermídia de larga escala.

A uniformização da interface do serviço faz com que os métodos utilizados para a interação do cliente com o servidor sejam simples. HTTP fornece, para a maioria das operações, quatro métodos comuns: GET, PUT, POST e DELETE (RICHARDSON, 2007).

Burke (2010), afirma que a interface restringida de uma maneira uniforme é importante para tornar as interações previsíveis, pois, como todas os recursos estão disponíveis por URIs, é possível identificar quais métodos podem estar disponíveis para a utilização dos recursos.

4.3. Funcionamento

Conforme abordado anteriormente, a REST e ROA tem como foco principal os recursos e como eles são representados no serviço.

A figura 15 apresenta uma visão mais completa do modelo arquitetural orientado a serviços. O recurso é definido como a parte central do modelo, ele pertence a um serviço (*person or organization*), possui URIs, descrição, representação e regras aplicadas. E o cliente (*agent*) pode descobri-lo de maneira intuitiva ou através de uma requisição a um serviço de descoberta.

Segundo Booth et al. (2004) um serviço de descoberta permite ao agente encontrar descrições dos recursos do serviço Web. Ele é utilizado para publicar e procurar as descrições dos recursos. Embora o modelo orientado a recursos seja de propósito geral, o principal recurso disponível no *Web services* é o próprio serviço.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

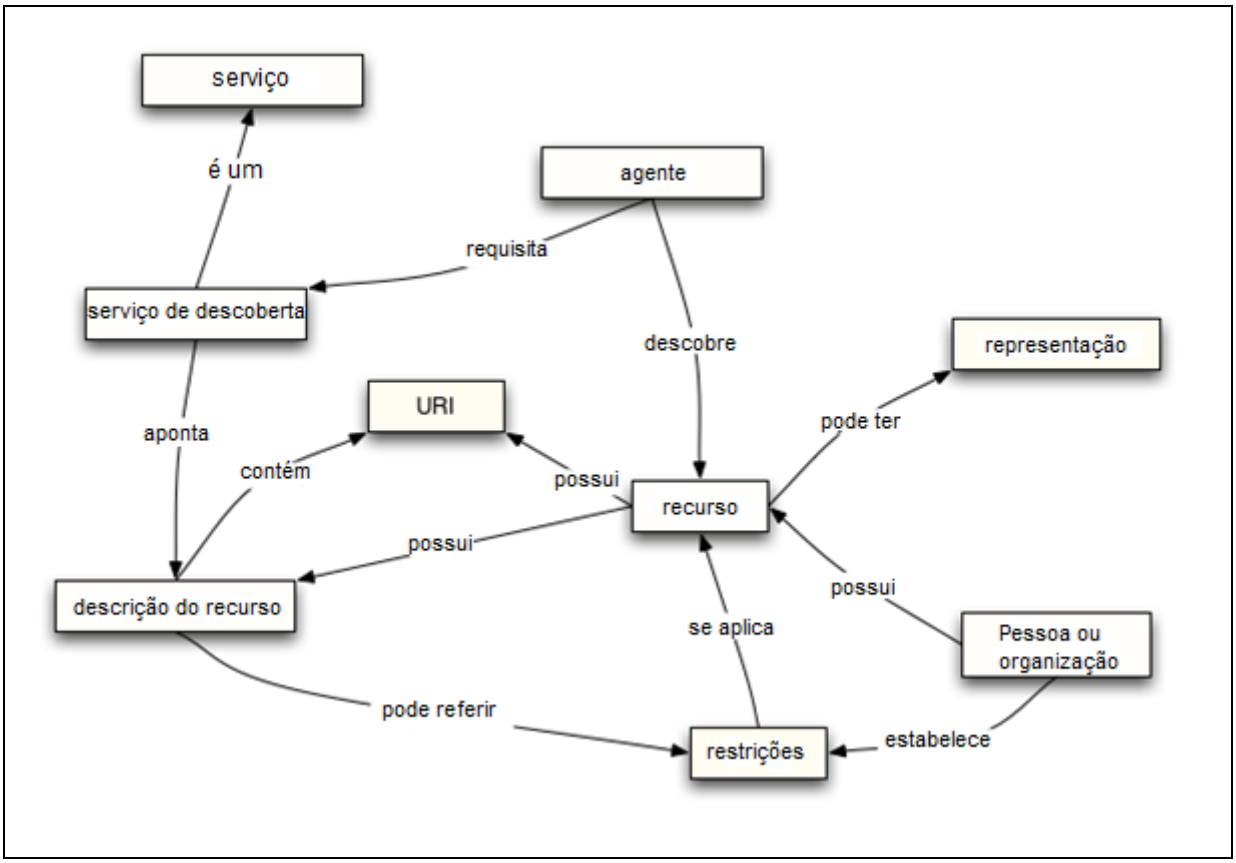

Figura 15 - Visão completa do modelo orientado a recursos. (In: Booth et al. 2004).

Todas as requisições feitas ao Web services são no formato de mensagens HTTP. Além de possuir os métodos que definem as requisições HTTP, também possui códigos que definem as respostas esperadas pelo serviço, caso uma requisição tenha sucesso ou não.

Segundo Richardson (2007), é possível fazer todas as operações dos Web services com os quatro principais métodos do HTTP. GET serve para recuperar uma informação ou simplesmente visualizar um resultado. O DELETE funciona para apagar uma informação no recurso. PUT cria um novo recurso, em um novo URI. É nesse momento que o estado da aplicação é transferido pelo serviço para se tornar o estado de um recurso. POST, assim como PUT, também cria um novo recurso, porém, neste último, o cliente que decide qual a URI do novo recurso. POST também é usado para modificar os recursos e criar recursos filhos.

Além desses quatro principais métodos, HTTP também possui HEAD, que funciona similarmente ao GET, mas apenas recupera os metadados (descrição) dos recursos.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

E o método OPTIONS, que checa quais métodos HTTP está disponíveis para cada recurso. (RICHARDSON, 2007)

Segundo Burke (2010), HTTP é bem consistente na especificação dos métodos e, para assegurar a comunicação entre cliente e servidor, ele possui um código ou uma numeração para cada tipo de resposta. As respostas que o serviços enviam após a requisição dos clientes podem ser de Sucesso ou Erro. Respostas de sucesso variam entre os códigos 200 e 399, enquanto as respostas de erro variam do código 400 e 599.

A figura 16 apresenta alguns dos principais códigos de resposta do HTTP e seus significados.

```
Status-Code    =
      "100"  ; Section 10.1.1: Continue
    | "101"  ; Section 10.1.2: Switching Protocols
    | "200"  ; Section 10.2.1: OK
    | "201"  ; Section 10.2.2: Created
    | "202"  ; Section 10.2.3: Accepted
    | "203"  ; Section 10.2.4: Non-Authoritative Information
    | "204"  ; Section 10.2.5: No Content
    | "205"  ; Section 10.2.6: Reset Content
    | "206"  ; Section 10.2.7: Partial Content
    | "300"  ; Section 10.3.1: Multiple Choices
    | "301"  ; Section 10.3.2: Moved Permanently
    | "302"  ; Section 10.3.3: Found
    | "303"  ; Section 10.3.4: See Other
    | "304"  ; Section 10.3.5: Not Modified
    | "305"  ; Section 10.3.6: Use Proxy
    | "307"  ; Section 10.3.8: Temporary Redirect
    | "400"  ; Section 10.4.1: Bad Request
    | "401"  ; Section 10.4.2: Unauthorized
    | "402"  ; Section 10.4.3: Payment Required
    | "403"  ; Section 10.4.4: Forbidden
    | "404"  ; Section 10.4.5: Not Found
    | "405"  ; Section 10.4.6: Method Not Allowed
    | "406"  ; Section 10.4.7: Not Acceptable
    | "407"  ; Section 10.4.8: Proxy Authentication Required
    | "408"  ; Section 10.4.9: Request Time-out
    | "409"  ; Section 10.4.10: Conflict
    | "410"  ; Section 10.4.11: Gone
    | "411"  ; Section 10.4.12: Length Required
    | "412"  ; Section 10.4.13: Precondition Failed
    | "413"  ; Section 10.4.14: Request Entity Too Large
    | "414"  ; Section 10.4.15: Request-URI Too Large
    | "415"  ; Section 10.4.16: Unsupported Media Type
    | "416"  ; Section 10.4.17: Requested range not satisfiable
    | "417"  ; Section 10.4.18: Expectation Failed
    | "500"  ; Section 10.5.1: Internal Server Error
    | "501"  ; Section 10.5.2: Not Implemented
    | "502"  ; Section 10.5.3: Bad Gateway
    | "503"  ; Section 10.5.4: Service Unavailable
    | "504"  ; Section 10.5.5: Gateway Time-out
    | "505"  ; Section 10.5.6: HTTP Version not supported
    | extension-code
```

Figura 16 - respostas HTTP (In: Fielding et al., 1999).

Segundo Burke (2010) uma característica do funcionamento de REST, que difere dos demais Web services, como os serviços baseados em SOA, é a restrição de HATEOAS (*Hypermedia As The Engine Of Application State*) ou Hipermídia como o motor do estado do aplicativo. Tal princípio define que uma aplicação cliente pode interagir com uma aplicação em rede através de hipermídias geradas automaticamente por um servidor de aplicação. Ou seja, um cliente REST não necessita de muitas informações para interagir com um serviço, o simples conhecimento de interação com as hipermídias já é suficiente. A restrição de HATEAOS permite uma grande separação entre o cliente e o *Web services* de forma que o serviço pode funcionar independente por completo.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

REST é uma tecnologia relativamente nova, que está sendo amplamente utilizada por grandes empresas atualmente. Sua arquitetura e princípios, definidos por *Roy Fielding,* em sua dissertação em 2001, apesar de parecerem restritivas se apresentaram bastantes eficientes em comparação ao cenário dominado pelo *Web services* baseados em SOAP e SOA (NEWCOMER, 2002).

Após serem definidas as principais tecnologias e modelos arquiteturais dos *Web services* e após a descrição do estilo arquitetural REST para serviços *Web*, o próximo capítulo apresentará o desenvolvimento de um *Web sevice* RESTful que seguirá os padrões já abordados. Também serão apresentadas aplicações para consumir tal serviço e mostrar seus possíveis resultados.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

5 – IMPLEMENTAÇÃO

Com base nos aspectos abordados ao longo deste trabalho, este capítulo visa apresentar a desenvolvimento de um *Web services* que segue os padrões REST. Tal serviço possuirá métodos que serão consumidos por aplicações clientes, além de implementar um módulo de autenticação e validação de *logins* de usuários através do *realm* de autenticação do servidor. O consumo do *Web services* será demonstrado por um *Web Browser* e por uma classe Java.

Segundo Burke (2010), o método de validação em *Web services* consiste em autenticação e autorização. A autenticação resume-se em validar a identidade de um cliente que tenta acessar determinado serviço. Normalmente, a autenticação consiste em checar se o cliente forneceu credenciais existentes como login e senha. O próprio *servlet* contêiner do JavaEE possui as bibliotecas necessárias para entender e configurar os protocolos de autenticação e segurança. Após a autenticação, é necessário checar a autorização do *login*, definindo assim, se o cliente pode ou não interagir com os recursos disponíveis na URI.

5.1 Especificações da Aplicação

Para desenvolvimento dessa aplicação será utilizado a linguagem de programação JavaEE por possuir uma grande coleção de bibliotecas e APIs para o desenvolvimento de Web services e aplicações web e proporcionando, também, bastante facilidade para a criação de serviços (recursos) e clientes através de classes e manipulação de objetos e métodos. A ferramenta de desenvolvimento será o NetBeansIDE 7.2.

Para a implantação da metodologia REST, restringida à linguagem de programação Java, foram utilizadas as bibliotecas do Jersey, que é uma implementação da especificação JAX-RS (JSR-311), que define a criação de um Web services descrita pelo estilo arquitetural RESTful. Utilizou-se, ainda, o servidor GlassFish Server Open

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

Source Edition 3.1.2 responsável pelo gerenciamento do serviço e do acesso dos clientes ao provedor de serviços.

5.1.1. Java

Graças à API de servlets, é possível escrever Web services RESTful em Java. Servlets possuem grande proximidade com o protocolo HTTP e possuem muitos trechos de códigos que podem ser reusados para gerenciar transações com o HTTP.

Apesar dos Web services usarem frameworks baseados em padrões para definir o escopo dos serviços, ainda é necessário que o serviço seja implementado em uma infraestrutura de aplicação. JavaEE define tal infraestrutura que suporta confiabilidade, segurança e eficácia. Dentro do J2EE, os componentes de servlets, JSPs e EJBs, drivers JDBC e adaptadores proveem acesso às regras de negócios e recursos que os Web services necessita. Dessa maneira, os servlets serão os pontos de entrada para os serviços e uma ponte automática entre uma mensagem de um Web services e qualquer outro serviço contido dentro de um servidor de aplicações.

5.1.2 NetBeans

Uma IDE (integrated development environment), ou ambiente de desenvolvimento integrado para desenvolvimento de sistemas na plataforma JavaEE, e outras linguagens de programação, tais como C, C++ e PHP. Pode ser encontrada gratuitamente para diversos Sistemas Operacionais, além de possibilitar o desenvolvimento de outras linguagens como em www.netbeans.org.

5.1.3. JAX-RS e Jersey

JAX-RS (*Java API for RESTful Web services*) é uma especificação criada em 2008 para simplificar a implementação de serviços RESTful, aplicando anotações em

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

Objetos Java. Tais anotações podem especificar os padrões das URIs e as operações HTTP para cada método da classe. Além de especificar anotações para injeção de parâmetros que facilita o acesso às informações das requisições HTTP. JAX-RS também possui leitores e escritores de mensagens que permitem dissociar dados dos Objetos de dados Java, permitindo assim, transformar esses dados em informação para serem enviados em transações entre clientes e serviços.

A figura 17 apresenta alguns exemplos de anotações utilizadas em uma classe com os métodos que definem o serviço. A anotação *@Path* define o caminho que será a URI do serviço, nesse caso "/autentica". A anotação *@GET* define qual será o tipo do método e a anotação. *@Produces* define o tipo de dados que o método irá enviar para uma resposta a um usuário. A anotação *@PUT* define o método que executará a instrução PUT do protocolo HTTP. Por fim, a anotação *@Consumes* define o tipo de dados que será recebido por ela para poder executar sua operação.

```
23  @Path("/autentica")
24  public class ServicoAutenticacao {
25
26    @GET
27    @Produces(MediaType.TEXT_PLAIN)
28    public String retornarTexto() {
29      return "Autenticação Bem sucedida";
30    }
31
32    @GET
33    @Path("/buscaraluno")
34    @Produces(MediaType.APPLICATION_XML)
35    public ArrayList<Aluno> selecionarTodos(){
36        return ManipularDados.getInstance().listarAlunos();
37    }
81    @PUT
82      @Consumes({MediaType.TEXT_PLAIN})
83      public void putMessage(String content) {
84          Logger.getLogger(ServicoAutenticacao.class.getName()).info(content);
85      }
```

Figura 17 - Exemplo de anotações Jersey.

Como JAX-RS é uma especificação, ele precisa ser implementado de alguma forma. Jersey é a implementação referência, de código aberto, do JAX-RS. Está sob as licenças CDDL(Common Developmente and Distribution Licence) versão 1.0 e GPL(GNU General Public License) versão 2.

Jersey é projetado, montado e instalado usando o Apache Maven. O projeto e suas especificações podem ser encontrados no site principal htt://jersey.java.net. Uma

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

vantagem do Jersey é que ele já está implantado no GlassFish Server OpenSource, e com isso alguns erros de integração são automaticamente eliminados.

5.1.4. GlassFish Server Open Souce

GlassFish Server Open Source Edition fornece um servidor para o desenvolvimento e implantação de aplicações JavaEE e tecnologias da Web baseadas na tecnologia Java.

Dentre as características do GlassFish Server Open Source podemos destacar a existência de um web container, uma interface para administração de configuração e gerenciamento dos recursos oferecidos, tornando bastante intuitiva a instalação de novos plugins e componentes.

O servidor também possui grande estabilidade para servlets e as demais tecnologias do JavaEE e já possui o Jersey configurado. Suas especificações, documentação e downloads podem ser encontrados no site http://glassfish.java.net.

5.2. Tabelas do Serviço

O *Web services* irá interagir com três tabelas, para consumir dados e para realizar a autenticação e validação dos *logins* dos usuários.

Os métodos de autenticação e autorização serão feitos pela web container do servidor de aplicação (GlassFish Open Source). Os usuários serão autenticados se suas credencias estiverem corretas e se o usuário pertencer ao grupo correspondente. Para que isso ocorra é necessária à criação de um banco de dados para o armazenamento das informações de logins, senhas e grupos.

A figura 18 exemplifica as definições das tabelas usadas para esse modelo.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

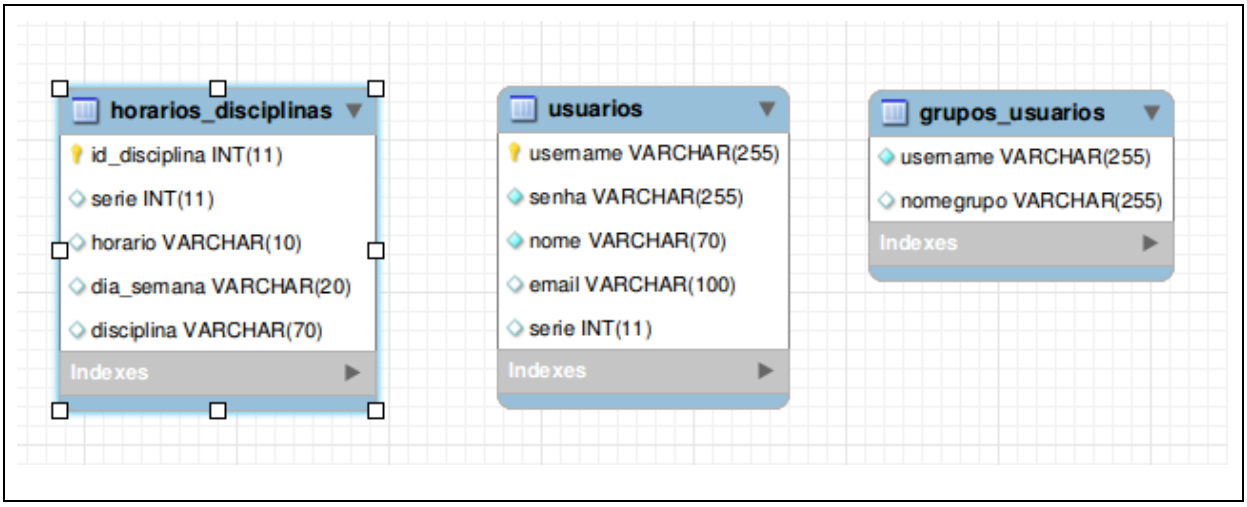

Figura 18 – Tabelas do banco de dados.

É importante ressaltar o motivo de definir o nome da coluna username para as duas tabelas, pois as duas colunas serão usadas para unir a tabela de usuários com a tabela de grupos, assim o realm de segurança do Glassfish executará a instrução, conforme figura 19 para recuperar os usuários e grupos, que definem os tipos de permissões para cada login:

```
1 SELECT nomegrupo FROM grupos_usuarios g, usuarios u
2  where g.username = u.username and u.username = ?
```

Figura 19 – Instrução para recuperar credenciais dos logins.

5.3. *Realm* de autenticação

Realm de autenticação, também conhecido como domínio de segurança é a definição de um escopo no qual o servidor de aplicação define suas regras comuns de segurança para autenticação e autorização de *logins* para qualquer aplicação.

A figura 20 demonstra a criação de um *realm* JDBC no Glassfish .

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

Figura 20 – Realm JDBC.

Conforme abordado na sessão anterior, só é possível proteger uma aplicação pela definição de usuários e grupos de segurança em um realm de segurança do servidor de aplicação. Dessa maneira, a aplicação pode especificar qual realm usar em seu descritor de implantação, no caso do JavaEE isso ocorre no arquivo web.xml, que define as configurações de autenticação como: o método utilizado, e qual os pacotes devem ser protegidos pelo servidor. O arquivo "web.xml" será descrito com mais detalhes na próxima sessão.

Existem alguns tipos de realms de segurança. Os mais comuns podem ser listados como: MemoryRealm – implementação simples que guarda os dados do administrador do servidor como um arquivo xml; JNDIRealm – implementação que utiliza a Java Naming and Directory Interface(JNDI); DataSourceRealm – Utiliza um datasource JDBC configurado por JNDI.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

Para recuperar as credencias de usuários salvas em um banco de dados é utilizado um realm JDBC. Nele o servidor usa as informações da base de dados e as opções descritas no arquivo web.xml.
Conforme abordado na sessão anterior, o próprio realm estará encarregado de fazer uma consulta nas tabelas do banco para verificar as credenciais dos logins. Cada usuário pertence a um grupo e cada grupo possui privilégios.

A figura 21 exemplifica os usuários e seus respectivos grupos que serão verificados pelo realm de segurança.

#	username	nomegrupo
1	teste1	usuario
2	teste2	semgrupo
3	teste3	admin
4	teste4	admin

Figura 21 – Grupos e usuários.

É importante ressaltar que no realm está definido um algoritmo de segurança SHA-256, isso indica que os usuários cadastrados no banco terão suas senha criptografadas. Porém, o realm fará a criptografia e decriptografia das credencias conforme o login dos usuários.

A figura 22 ilustra os usuários e suas senhas já criptografadas no algoritmo SHA-256

#	username	senha	nome	email	serie
1	teste1	15bf532d22345576b4a51b96da4754c039ef3458494066d76828e893d69ebd1e	Adriel	adrielcavaleiro@gmail.com	4
2	teste2	f91a799a01ebac97c8995f50381b5be0d43ca7eb0c6a6c6bc07603c5a922d1fa	SemNome	semnome@semnome.com	4
3	teste3	ee80b6f59a0e2c1c30853d1be6246defa9e70c10f8efb2178cabe14f2edca6ba	teste3	teste3@teste3.com	3
4	teste4	09c45dae0b27fc48e3354639df145d59e5ba32a5e32fcedc59bf77e9cfa8f99e	teste4	adrielcavaleiro@gmail.com	4

Figura 22 - Usuários criptografados.

5.4. Serviço

O serviço utilizado para demonstrar as especificações e teorias abordadas ao longo deste trabalho possui uma estrutura simples, dividida em camadas, para acesso ao banco de dados, manipulação de e criação objetos e a camada onde está definido o

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

Web services e as URIs de seus métodos, para autenticação e validação de logins e acesso à outras informações.

A figura 23 apresenta a estrutura geral da aplicação desenvolvida.

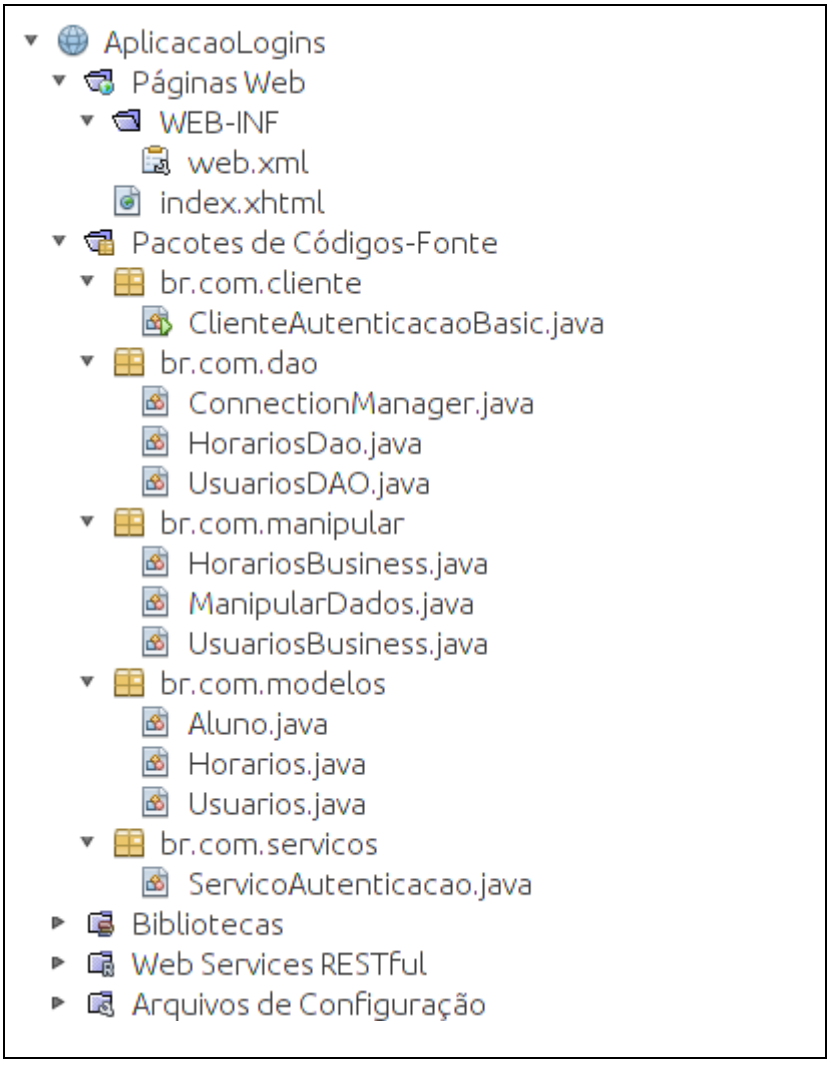

Figura 23 – Estrutura Geral da Aplicação.

A definição do Web services possui uma estrutura simples contendo métodos HTTP GET para retornar diferentes mensagens em diferentes formatos e um método PUT, responsável pelo recebimento das credenciais para a autenticação do cliente.

Cada método possui um tipo de dados diferente para enviar para cada tipo de cliente. Por exemplo, se um web browser acessar o serviço o método de retorno será por padrão retornarHtml(). Em outras aplicações podem ser usados os métodos retornarTexto() ou retornarXML() que enviam, respectivamente, mensagens em formato de uma String de texto ou em um documento XML.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

As figuras 24 e 25, a seguir, apresentam a classe principal do serviço e a definição de seus métodos

```
@Path("/autentica")
public class ServicoAutenticacao {

  @GET
  @Produces(MediaType.TEXT_PLAIN)
  public String retornarTexto() {
    return "Autenticação Bem sucedida";
  }

  @GET
  @Path("/buscaraluno")
  @Produces(MediaType.APPLICATION_JSON)
  public ArrayList<Aluno> selecionarTodos(){
      return ManipularDados.getInstance().listarAlunos();
  }

  @GET
  @Path("/buscarusuario")
  @Produces(MediaType.APPLICATION_JSON)
  public Usuarios selecionarUsuarios(){
          Usuarios usuario = new UsuariosBusiness().buscarUsuario();
          return usuario;
  }

  @GET
  @Path("/listarUsuario")
  @Produces(MediaType.APPLICATION_XML)
  public List<Usuarios> listartodos(){
      List<Usuarios> lista = new UsuariosBusiness().buscarTodos();
      return lista;
  }
```

Figura 24 – Classe do Serviço parte 1.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

```
    @GET
    @Path("/listarHorarios")
    @Produces(MediaType.APPLICATION_JSON)
    public List<Horarios> listarHorarios(){
        List<Horarios> lista = new HorariosBusiness().buscarTodos();
        return lista;
    }

    @GET
    @Produces(MediaType.TEXT_XML)
    public String retornarXML() {
      return "<?xml version=\"1.0\"?>" + "<inicio> Autenticacao Com Sucesso" + "</inicio>";
    }

    @GET
    @Produces(MediaType.TEXT_HTML)
    public String retornarHtml() {
      return "<html> <body> <div id='container' style='width:100%;float:center;'> "
              + "<div id='header' style='background-color:#FFA500;'>"
              + " <h1 style='margin-bottom:0;text-align:center;'>Login Autenticado Com Sucesso</h1></div>"
              + "<div id='content' "
              + "style='background-color:#EEEEEE;height:70%;width:100%;text-align:center;'> "
              + "Este M&eacutetodo s&oacute &eacute mostrado em requests HTML</div> "
              + "<div id='footer' style='background-color:#FFA500;clear:both;text-align:center;'> "
              + "GET retornando HTML</div> </div> </body> </html>";
    }

    @PUT
      @Consumes({MediaType.TEXT_PLAIN})
      public void putMessage(String content) {
          Logger.getLogger(ServicoAutenticacao.class.getName()).info(content);
      }

  }
```

Figura 25 – Classe do Serviço parte 2.

Para que o serviço possa funcionar corretamente é necessário descrever o servlet do Jersey onde se encontra o pacote de inicialização do serviço e o padrão de URIs que será utilizado pelo mesmo. A figura 26, a apresenta a definição do servlet do Jersey no arquivo web.xml.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

```
<servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
        <param-name>com.sun.jersey.config.property.packages</param-name>
        <param-value>br.com.servicos</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
<session-config>
    <session-timeout>
        30
    </session-timeout>
</session-config>

<welcome-file-list>
    <welcome-file>index.faces</welcome-file>
</welcome-file-list>
```

Figura 26 – Servlet Jersey.

É também dentro do documento web.xml, conforme mencionado na sessão anterior, que estão definidos as regras de autenticação do serviço e o realm que o serviço irá utilizar para completar as regras de segurança. Conforme ilustrado na figura 27, é possível identificar o realm que a aplicação está usando, assim como o tipo de usuários que terá privilégio para acessar os métodos do recurso.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

```
<security-constraint>
    <display-name>Basic Protection</display-name>
    <web-resource-collection>
        <web-resource-name>REST</web-resource-name>
        <description/>
        <url-pattern>/rest/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <description/>
        <role-name>admin</role-name>
    </auth-constraint>
</security-constraint>

<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>myRealm</realm-name>
</login-config>
<security-role>
    <description>Users</description>
    <role-name>admin</role-name>
</security-role>
```

Figura 27 – Regras de segurança web.xml.

5.5. Clientes para Teste

As próximas sub sessões descreverão os clientes para o teste do Web services.
Para testar a aplicação foram utilizados dois tipos de clientes: um Web Browser contendo páginas html e uma classe Java de teste para a autenticação.

5.5.1. Web Browser.

Utilizando um Web browser como cliente para acessar o serviço, foram encontrados os seguintes resultados.

A figura 28 demonstra a tela inicial no momento em que o browser requisita a autenticação.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

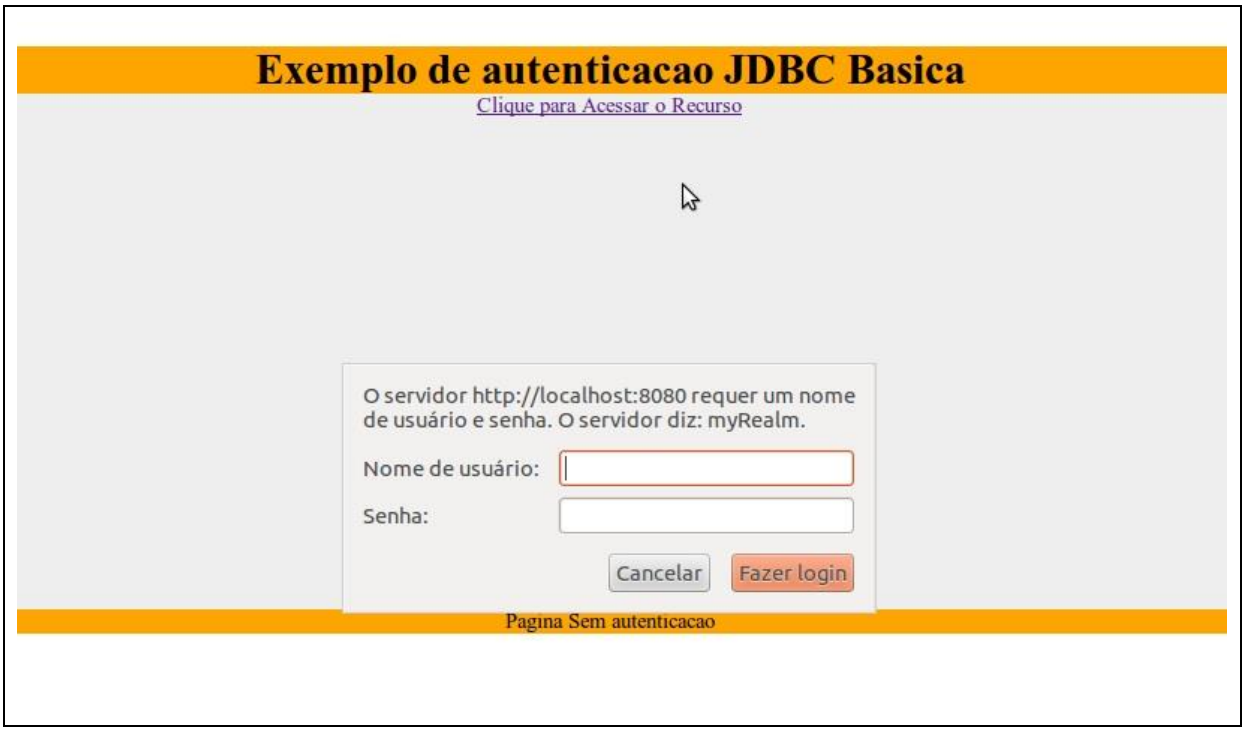

Figura 28 – Requisição de Usuario e Senha.

A figura 29 apresenta a resposta de um login existente, porém não autorizado para o acesso ao recurso. A resposta é definida pelo código 403 do protocolo HTTP. Nesse exemplo foi utilizado o login "teste1" armazenado na tabela "usuarios" do banco de dados da aplicação.

Figura 29 – Erro 403 não autorizado.

A figura 30 ilustra a resposta de um login não existente e que também não possui autorização para acessar o recurso. A resposta é definida pelo código 401 do protocolo HTTP.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

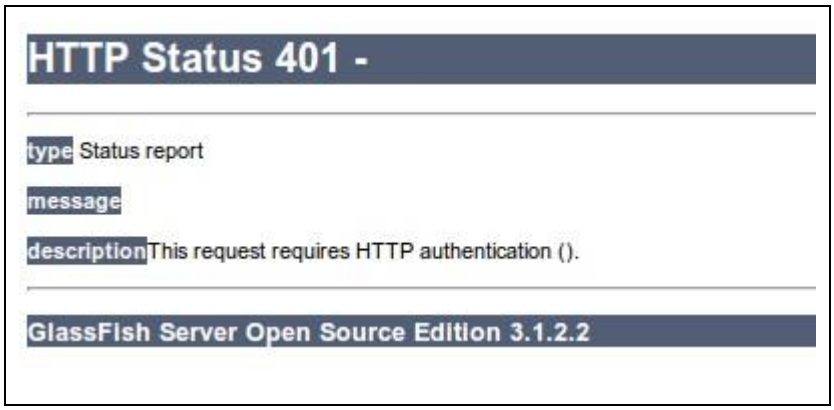

Figura 30 – Erro 401 – usuário inexistente.

A figura 31 apresenta o resultado de um cliente autorizado e autenticado pelo serviço, e a resposta do método GET que retornou um documento HTML. Nesse exemplo o código de resposta da requisição HTTP seria 200 – OK.

Figura 31 – usuário autenticado.

Além da autenticação e autorização dos usuários o Web services possui alguns métodos para consumo de aplicações clientes. Os métodos definidos como "selecionarUsuarios()", "listarTodos()" e "listarHorarios()" possuem URIs diferentes

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

da URI principal utilizada para a autenticação ("/autentica"), e cada método faz dissociação dos objetos Java, transformando-os em arquivos XML ou JSON.

Isso torna mais claro a importância e a utilização do XML com as tecnologias dos Web services, por ser mais fácil transportar um objeto Java, recuperado de um banco de dados, no formato de um documento XML ou JSON por, principalmente ocupar menos volume de dados.

A figura 32 apresenta o documento no formato JSON formado como resultado do método "selecionarUsuarios()" através da URI "/buscarusuario".

Figura 32 – Buscar Usuário.

A figura 33 apresenta o documento no formato XML formado como resultado do método "listarTodos()" através da URI "/listarUsuario".

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

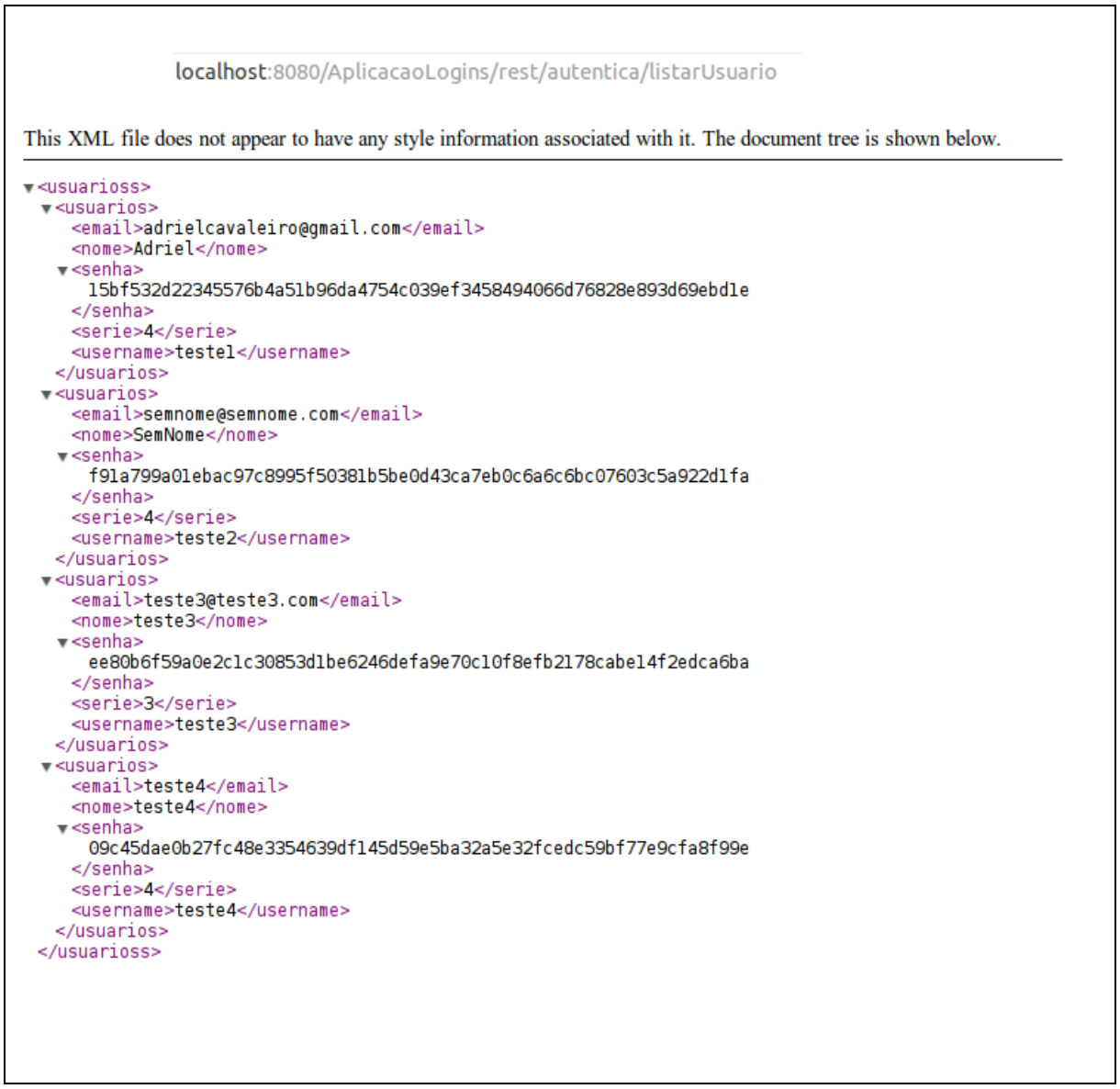

Figura 33 – Listar Usuario.

5.5.2. Cliente Java

Utilizando uma classe Java com método principal como cliente os resultados encontrados foram os mesmos aos encontrados no web browser com diferença apenas na resposta do serviço, que pôde ser escolhida pelo cliente ao invés de receber um documento HTML já definido como ocorreu no browser. No caso da Classe Java também é possível ver como as mensagens são enviadas e recebidas com mais detalhes.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

A figura 34 demonstra a classe Java com seus métodos para realizar a autenticação do usuário e imprimir a resposta em um console.

```
19  public class ClienteAutenticacaoBasic {
20
21      public static void main(String[] args) {
22          ExemploCliente exc = new ExemploCliente();
23          exc.setUsernamePassword("teste4", "teste4");
24          System.out.println(exc.getMessageXML());
25          System.out.println(exc.getMessageStr());
26          System.out.println(exc.getMessageHTML());
27
28          exc.close();
29
30      }
31
32      ////////////////////////////////////////////////////////////////////////////////
33      static class ExemploCliente {
34
35          private Client client;
36          private WebResource webResource;
37          private static final String BASE_URI = "http://localhost:8080/AplicacaoLogins/rest";
38
39          public ExemploCliente() {
40              ClientConfig configClient = new DefaultClientConfig();
41              client = Client.create(configClient);
42              client.addFilter(new LoggingFilter());
43              webResource = client.resource(BASE_URI).path("autentica");
44          }
45
46          public String getMessageXML() throws UniformInterfaceException {
47              WebResource resource = webResource;
48              return resource.accept(MediaType.TEXT_XML).get(String.class);
49          }
50
51          public String getMessageHTML() throws UniformInterfaceException {
52              WebResource resource = webResource;
53              return resource.accept(MediaType.TEXT_HTML).get(String.class);
54          }
55
56          public String getMessageStr() throws UniformInterfaceException {
57              WebResource resource = webResource;
58              return resource.accept(MediaType.TEXT_PLAIN).get(String.class);
59          }
60
61          public void close() {
62              client.destroy();
63          }
64
65          public void setUsernamePassword(String username, String password) {
66              client.addFilter(new HTTPBasicAuthFilter(username, password));
67          }
68      }
69  }
```

Figura 34 – Cliente Java.

Nessa classe existe uma classe estática chamada ExemploCliente. Os objetos criados nela como client e webResource são definidos de classes do Jersey do mesmo nome. O objeto client define, a partir do método "setUsernamePassword" as credenciais do usuário que irá acessar o recurso descrito na String BASE_URI. Após

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

definidos o login e senha do client, o objeto webresource se encarrega de fazer a chamada ao *Web services* utilizando o cliente cadastrado, a uri principal(BASE_URI) do serviço e o caminho da URI de autenticação("/autentica"), e assim ele pode acessar os métodos "getMessageXML()", "getMessageHTML()" e "getMessageStr()" para mostrar a resposta do serviço no terminal.

A figura 35 apresenta a resposta no console da aplicação para um cliente existente e autorizado.

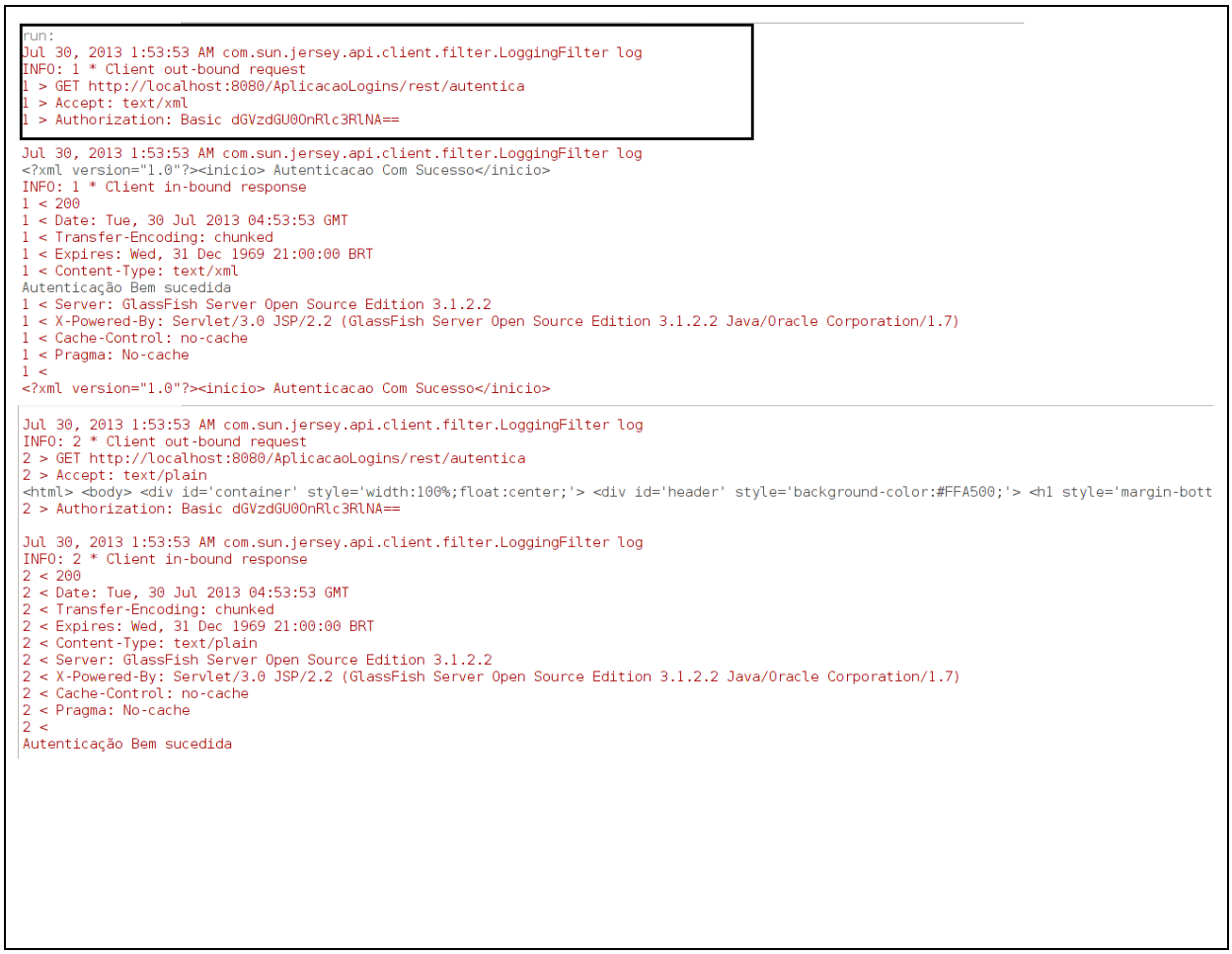

Figura 35 – Cliente Java Autorizado.

Na Imagem acima o cliente foi autorizado e a aplicação imprimiu no terminal os três tipos de resposta da autenticação: um texto no formato de XML, um texto simples e um texto formatado como HTML. É importante comentar sobre a área demarcada dessa figura, antes do cliente fazer a autenticação no *Web services* já havia enviado uma mensagem contendo o tipo de autenticação utilizado(Basic) e o tipo de dados

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

que ele aceitaria (text/xml). Após a autenticação o retornou uma mensagem que continha o código 200 do HTTP – "Ok".

A figura 36 apresenta a resposta no console da aplicação para um cliente existente, porém não autorizado. Nesse caso, similar à aplicação com Web Browser, o cliente utilizado foi "teste1". O código do erro está descrito no segundo bloco do console indicando erro 403 – não autorizado.

Figura 36 – Cliente Java não autorizado, erro 403.

A figura 37 apresenta a resposta no console da aplicação para um cliente inexistente. Nesse caso as *Strings* que definem usuário e senha foram definidas em branco. O código apresentado foi 401 referente ao usuário não existente.

Figura 37 – Cliente Java inexistente- erro 401.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

CONSIDERAÇOES FINAIS

O trabalho proposto apresentou, através do levantamento bibliográfico, um estudo para determinar Web Services. Estes podem ser considerados sistemas de software designados para suportar iterações interoperáveis de máquina-para-máquina sobre uma rede de computadores. Outros sistemas interagem com o *Web Services* numa maneira pré-estabelecida, pelo próprio serviço, utilizando mensagens tipicamente transmitidas através do protocolo HTTP com uma serialização XML em conjunto com outros padrões da Web

Foi também observada a arquitetura geral dos Web Services e seus modelos arquiteturais existentes, dando destaque ao modelo arquitetural orientado à recursos, base do estilo arquitetural REST. Também foram abordadas as principais tecnologias utilizadas para a criação de serviços web como SOAP, HTTP e XML.

Das tecnologias apresentadas foi feito um estudo sobre o XML, considerado a "espinha dorsal" dos Web Services e também uma das tecnologias mais presentes na Internet. Foi destacada sua importância como uma linguagem de marcação criada para gerar e estruturar dados na forma de documentos, tanto para a definição de arquivos de representação de baixo nível para descrever, por exemplo, arquivos de configuração, quanto para um meio de adicionar meta-dados a documentos, similar ao HMTL e assim gerar uma das especificações que definem Web Services (WSDL). Além disso, XML é a tecnologia utilizada para definir a estrutura das mensagens dos Web Services tradicionais, e no caso dos Web Services RESTful, pode ser usado nas mensagens para estruturar as representações trocadas pelos serviços e clientes.

Após o estudo das principais tecnologias dos Web Services, foi observado REST, como um estilo arquitetural para criação de aplicações distribuídas de hipermídia interconectadas por hiperlinks e que podem ser transferidas entre agentes. A própria Internet é o melhor exemplo desse sistema. Como a principal característica dos Web

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

Services RESTful é assegurar que as aplicações sejam mais próximas e parecidas com a Internet propriamente dita, eles utilizam o protocolo HTTP como parte principal e fundamental de sua arquitetura. Dessa forma, os sistemas computacionais podem ser mais escaláveis, de fácil modificação e com performance melhorada para transações

Depois de feito o levantamento bibliográfico, foi implementado uma aplicação para demonstrar com sucesso os assuntos abordados, permitindo a definição de um serviço REST baseado nos padrões arquiteturais que o definem.

O serviço apresenta a autenticação e autorização de Logins feita no servidor Glassfish através da definição de regras de segurança e métodos para tornar possível a comunicação entre o cliente e o serviço.

Além da autenticação foi possível demonstrar as trocas de mensagens, estruturação dos dados e o acesso às informações pelos métodos do Web Service. Graças à portabilidade e consistência da linguagem de programação Java, foi possível a criação de um web Service RESTful utilizando o as bibliotecas Jersey da especificação do JAX-RS.

Foi através do Jersey que tornou-se possível uma rápida e simples implementação do serviço de autenticação e na utilização do serviço para consumo de aplicações clientes utilizando uma interface uniforme conforme descrita pelo modelo REST.

Para trabalhos futuros seria interessante o estudo sobre a utilização das anotações do JAX-RS para definir a segurança de cada método do recurso, deferindo de como foi apresentado nesse trabalho em que o recurso por completo foi restringido, dessa forma logins podem ter, ou não acesso, somente, à alguns métodos . Outro trabalho futuro proposto é a utilização de diferentes tipos de dados como arquivos de imagens, músicas ou vídeos para serem recuperados através de Stream de dados pelo cliente ou enviados para o serviço.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

Referências Bibliográficas

Booth, David et al. Web Service Architecture. W3C Working Group Note. Disponível em <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#introduction>. Acesso em: 5 Março 2013.

BURKE, William. **RESTful Java with JAX-RS**, First Edition. Gravenstein Higway North: O'Reilly, 2009.

CERAMI, Ethan, **Web Services Essentials ,** First Edition. Gravenstein Higway North: O'Reilly, 2002

CHAPELL, David; Tyler Jewell. **Java Web Services**, First Edition. Gravenstein Highway North: O'Reilly, 2002.

FAWCETT, Joe. **Beginning XML**, Fifth edition Indianapolis, Indiana: John Wiley & Sons, Inc. 2012

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

FIELDING, Roy Thomas et al. Hypertext Transfer Protocol. Disponível em: <http://www.w3.org/Protocols/rfc2616/rfc2616.html> Acesso em: 15 Fevereiro 2013.

GURUGE, Anura. **Web Services: Theory and Practice** , First Edition. ELSEVIER Digital Press, 2004

KALIM, Martin. Java Web Services: **Up and Running**, First Edition. Gravenstein Highway North: O'Reilly, 2009.

NEWCOMER, Eric. **Understanding Web Services – XML, WSDL, SOAP and UDDI**. Indianapolis: Pearson Education Corporate Sales Division, 2002.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

ORACLE. JAVA EE TUTORIAL: What Are RESTful Web Services? Disponivel em http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html. Acesso em: 25 Fevereiro 2013.

RICHARDSON, Leonard; RUBY, Sam. **RESTful Web Services**, First Edition. Gravenstein Highway North: O'Reilly, 2007.

SNELL, James; Kulchenco Pavel; Tidwell, Doug. **Programming Web Services with SOAP**, Fisrt Edition. Graveinstein Higway North: O'Reilly, 2001.

VLIST, Eric Van der. **XML Schema. First Edition,** Gravenstein Highway North: O'Reilly, 2002.

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br