FUNDAÇÃO DE ENSINO “EURÍPEDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURIPEDES DE MARÍLIA” – UNIVEM
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANTHONY FERREIRA LA MARCA

DIRETRIZES PARA DESENVOLVIMENTO DE APLICAÇÕES COM SERVIÇOS WEB

MARÍLIA
2008

ANTHONY FERREIRA LA MARCA

DIRETRIZES PARA DESENVOLVIMENTO DE APLICAÇÕES COM SERVIÇOS WEB

Trabalho de Curso apresentado ao Curso de Ciência da Computação da Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador:
Prof. Dr. VALTER VIEIRA DE CAMARGO

MARÍLIA
2008

LA MARCA, Anthony Ferreira
Diretrizes para desenvolvimento de aplicações com Serviço Web / Anthony Ferreira La Marca; orientador: Valter Vieira de Camargo, SP: [s.n], 2008.
55f.

Trabalho de Curso (Graduação em Ciência da Computação) – Curso Ciência da Computação, Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília - UNIVEM, Marília, 2005.

1. Diretrizes 2. Desenvolvimento 3. Serviço Web

CDD: 004.678

ANTHONY FERREIRA LA MARCA

DIRETRIZES PARA DESENVOLVIMENTO DE APLICAÇÕES COM SERVIÇO WEB

Banca Examinadora da monografia apresentada ao Curso de Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Grau de Bacharel em Ciência da Computação.

Resultado: 6,0 (Seis)

ORIENTADOR: _____
Prof. Dr. Valter Vieira de Camargo

1º EXAMINADOR: _____
Prof. Dr. Edmundo Sérgio Spoto

2º EXAMINADOR: _____
Prof. Mauricio Duarte

Marília, 13 de novembro de 2008.

LA MARCA, Anthony Ferreira. **Diretrizes para Desenvolvimento de Aplicações com Serviços Web**. 2008. 53 f. Trabalho de Curso (Bacharelado em Ciência da Computação) – Centro Universitário de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2008.

RESUMO

Com a velocidade em que surgem aplicações computacionais, torna-se importante o uso de tutoriais didáticos, criativos e bem detalhados. Baseando-se nisto, o propósito deste trabalho é gerar um Serviço Web que realize inserção, remoção e busca em uma agenda, utilizando a plataforma *JAVA,* a linguagem *XML* (*Extensible Markup Language*), o protocolo *SOAP* (*Single Object Acess Protocol*) e o protocolo *HTTP* (*Hypertext Transfer Protocol*). Existe também a *UDDI* (*Univers*al *Description Dicovery Interface*), um diretório geral onde é registrado o WSDL para posteriormente ser buscado e localizado os seus respectivos serviços por organizações interessadas pelos mesmos. O software escolhido para o desenvolvimento é um software de fácil compreensão, pois o objetivo do trabalho é fornecer um bom ponto de partida para o desenvolvimento de outros tutoriais relacionados. Para o seu desenvolvimento são utilizadas as seguintes ferramentas: *Net Beans* 6.0.1, *Tomcat*, *MYSQL* e o *framework Axis*, ferramenta responsável por gerar toda a interface do Serviço Web, o descritor *WSDL.*

Palavras-chave: Serviços Web, *JAVA, XML, WSDL, UDDI, HTTP, Net Beans, Tomcat, MYSQL.*

LISTA DE ILUSTRAÇÕES

Figura 1.1 – Arquitetura de um Serviço Web.....10
Figura 1.2 - Aplicação Cliente acessando diretamente um Serviço Web.....10
Figura 1.3 - Registro UDDI.....14
Figura 1.4 - Pilha de Comunicação de um Serviço Web.....16
Figura 1.5 - SOAP sem anexos.....16
Figura 1.6 - Troca de mensagem One-Way e Request/Reply.....18
Figura 3.1 – Criação da tabela contato.....24
Figura 3.2 – Conexão.java.....25
Figura 3.3 – ContatoDAO.java.....26
Figura 3.4 – Criando o JAR.....28
Figura 3.5 – Tomcat 28
Figura 3.6 – Diretório lib do Axis 29
Figura 3.7 – Apache Axis 30
Figura 3.8 – Validate Apache Axis 30
Figura 3.9 – Serviço.java.....31
Figura 3.10 – WSDL Servidor 32
Figura 3.11 – Código WSDL Servidor 33
Figura 3.12 – Teste Cliente-Servidor.....35
Figura 3.13 – Cliente_Inserção.....36
Figura 3.14 – Resultado Inserção.....37
Figura 3.15 – Cliente_Buscar.....37
Figura 3.16 – Resultado da Busca.....38
Figura 3.17 – Cliente_Deletar.....39
Figura 3.18 – Resultado da Remoção.....39

LISTA DE ABREVIATURAS E SIGLAS

B2B: Business to Bussiness;

HTML: Hypertext Markup Language;

HTTP: Hypertext Transfer Protocol;

JAX-WS: Java API for XML Web Services;

JWS: Java Web Service;

REST: Representational State Transfer;

RPC: Remote Procedural Call;

SGML: Standard Generalized Markup Language;

SOAP: Single Object Acess Protocol;

UDDI: Universal Description Discovery and Integration;

URL: Uniform Resource Locator;

XML: Extended Markup Language;

W3C: World Wide Web Consortium;

WSDD: Web Service Deployment Descriptor;

WSDL: Web Services Description Language;

WSS4J: Web Service Security for Java;

SUMÁRIO

INTRODUÇÂO..... 7
Contexto..... 7
Motivação..... 8
Objetivo..... 8

CAPÍTULO 1 - SERVIÇOS WEB 9
1.1 Definição..... 9
1.2 XML..... 11
1.3 WSDL..... 12
1.4 UDDI.....13
1.5 SOAP..... 15
1.6 Aplicações..... 18

CAPÍTULO 2 - TRABALHOS RELACIONADOS..... 20

CAPÍTULO 3 - TUTORIAL SERVIÇO WEB..... 21
3.1 Projeto Proposto..... 23
3.2 Preparando Funcionalidades da Agenda..... 24
3.3 Preparando o Ambiente..... 28
3.4 O Servidor..... 31
3.5 O Cliente..... 34

CONCLUSÕES..... 41
REFERÊNCIAS..... 42
APÊNDICE..... 45

INTRODUÇÃO

Contexto

Um Tutorial é um conteúdo organizado e estruturado em formato de um documento ou programa, contendo imagens ou não, que tem por objetivo a aprendizagem, baseando-se num modelo auto-instrucional e na interação desse conteúdo (AOMESTRE, 2006).

Os tutoriais são utilizados na computação, pois a velocidade em que aplicações aparecem é muito alta e faz com que haja a necessidade de se criar um tutorial didático e bem detalhado.

Um tutorial voltado à computação possui potencialidades e características de comunicação e manipulação de informações muito interativas, tornando o processo de ensino-aprendizagem inovador, dinâmico, participativo e interativo.

Quando se trata da elaboração de um tutorial, devem ser levados em consideração dois aspectos fundamentais, a organização e a estética, que juntos representam "o cimento de uma boa construção". Estética é basicamente a aparência do tutorial, ou seja, se ele proporciona uma boa aparência e facilita a leitura do usuário por apresentar imagens, citações e cores, deixa o tutorial mais interessante de ser lido pelos mesmos. Já a organização consiste na forma que o tutorial se encontra, se ele está bem dividido e estruturado, se possui tópicos separados, bem escritos, organizados, sem erros, se possui parágrafos bem divididos para proporcionar uma boa leitura, enfim, tudo que deixa o usuário "preso" ao conteúdo.

Pode-se perceber que Estética e Organização tem o mesmo objetivo, atrair a atenção de leitores, isto é, ambos andam lado a lado para que um tutorial seja bem elaborado e estruturado ocasionando uma interação agradável com o usuário.

Já existem muitos tutoriais de Serviços Web (*Web Services*) disponibilizados na Internet, implementados em várias linguagens e plataformas. Também são encontrados em livros e até mesmo em artigos, (ANDERSON, 2008), (ABINADER, LINS, 2006).

Apesar da existência de muitos tutoriais sobre Serviços Web, conforme citado acima, percebe-se uma carência na literatura por tutoriais que sejam bem organizados e que abordem tópicos mais avançados com relação a este assunto. Sendo assim, esta monografia tem como objetivo apresentar uma proposta de criação de um tutorial para Serviços Web. O tutorial é elaborado de maneira fácil e didática para uma boa interpretação e interação com o usuário.

Motivação

Serviços Web é uma tecnologia “recente” que aborda um conceito novo de interoperabilidade, apresentando-se como uma nova alternativa na construção de aplicações para a Internet, oferecendo baixo acoplamento, possibilidade de escolha entre servidores e independência entre plataforma e segurança [W3C, 2002].

Conforme citado na Seção 1.1, já existem muitos tutoriais disponíveis na Internet, porém um “grande número apresenta deficiências e não abordam as principais características desse assunto” e, portanto, o tutorial apresentado será um tutorial simples que cumprira estes requisitos de maneira didática e objetiva.

Objetivo

Este trabalho tem objetivos principais e secundários. O objetivo principal é aumentar a facilidade de ensino de Serviços Web para que usuários e engenheiros de softwares possam aprender este novo tópico de maneira mais facilitada.

Como objetivos secundários podem-se citar a criação de um tutorial para apoiar o objetivo principal citado acima; a tarefa de abordar os tópicos mais relevantes relacionados com Serviços Web, como por exemplo, interoperabilidade e baixo acoplamento; fornecer facilidade de interação com o tutorial por meio de interfaces intuitivas e com bons níveis de usabilidade; fornecer um exemplo completo de desenvolvimento de uma aplicação por meio do tutorial.

CAPÍTULO 1 – SERVIÇOS WEB - *WEB SERVICE*

1.1-Definição

Quando surgiu a idéia de expandir a Internet, era mais do que suficiente à linguagem HTML (*Hypertext Markup Language*) e o protocolo HTTP (*Hypertext Transfer Protocol*), pois o interesse daquela época era apenas navegar em sites e usufruir seus conteúdos (ABINADER, 2006). A partir de então, criou-se a necessidade de ter aplicações mais complexas, como automatização de sistemas de empresas para atender às novas exigências do mercado.

As arquiteturas tradicionais se mostraram muito frágeis em relação a oferecer serviços para a Internet, pelo fato de terem um alto acoplamento entre vários componentes do sistema. A alta freqüência de mudança da Internet faz com que sistemas não consigam acompanhá-las, tornando-os instáveis e sujeito à falhas por serem altamente dependentes das partes que o compõe. A partir disso, houve a necessidade de se criar um serviço de aplicação que não fosse centralizado e também pudesse ser dinâmico a padrões da Internet.

Uma solução para estas questões foi o surgimento de uma nova tecnologia capaz de integrar sistemas com plataformas, ferramentas e linguagens distintas, ou seja, os Serviços Web. Os Serviços Web é um componente ou unidade de software que tem por objetivo integrar sistemas distintos por meio da Internet usando protocolos (no caso SOAP - *Single Object Acess Protocol* e HTTP) padronizados que garantem a sua independência de plataforma e linguagem.

Os Serviços Web possuem as seguintes características:

-uma forma comum de representar dados (XML - *Extended Markup Language*);

-um formato de mensagem comum e extensível (WSDL - *Web Services Description Language*);

-uma linguagem de descrição do serviço, comum e extensível;

-um mecanismo para localizar os serviços localizados em um Web site específico (UDDI - *Universal Description Discovery and Integration*);

-e um mecanismo para descobrir os provedores de serviços.

Uma arquitetura típica de um Serviço Web pode ser vista na Figura 1.1.

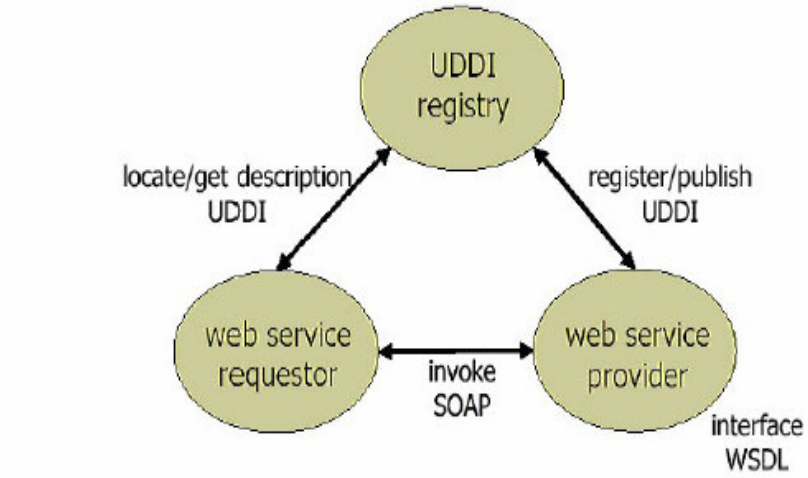

Figura 1.1: Arquitetura de um Serviço Web (OLIVEIRA, 2006)

Esta arquitetura consiste em três entidades:

-*Web Service Provider*: criam os Serviços Web e os publicam ao mundo externo registrando os serviços no *UDDI Registry*;

-*UDDI Registry*: mantém os registros dos serviços publicados;

-*Web Service Requestor*: após encontrar o serviço desejado, faz a conexão com o *Web Service Provider* por meio da interface WSDL e do protocolo SOAP para utilizarem o serviço.

Na Figura 1.2 é mostrada uma visão geral de como este processo ocorre:

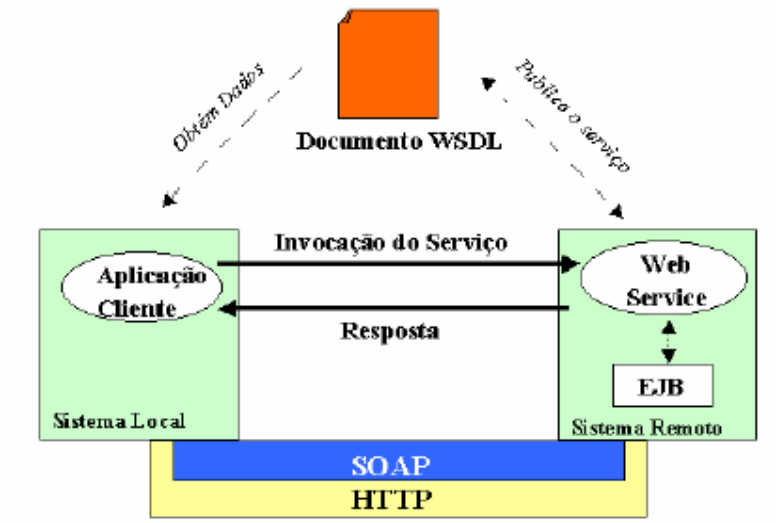

Figura 1.2 – Aplicação Cliente acessando diretamente um Serviço Web. (CUNHA, 2002)

A aplicação cliente ao localizar o serviço remotamente (por meio de um documento WSDL), invoca seus serviços por meio de uma RPC (*Remote Procedural Call*) e o Serviço Web estabelece a conexão, processa a chamada e envia a resposta pelo protocolo SOAP e HTTP, conforme ilustrada na Figura 1.2.

Os Serviços Web podem ser usados por uma ou várias aplicações por uma mesma organização, caso queira a integração dos sistemas da organização via Internet, basta que entenda SOAP e XML.

Para aplicações entenderem de maneira dinâmica a interface de Serviços Web, existe a linguagem WSDL, que permite publicar sobre a sintaxe, métodos, parâmetros e localização dos serviços.

Esses documentos podem ser armazenados no UDDI que funciona como um diretório geral, a partir dele, pode-se obter informações de um determinado serviço.

Apesar de todas as vantagens citadas acima, deve-se levar em consideração que os Serviços Web também sofrem um tipo de limitação, isto é, por estarem altamente relacionados com o ambiente no qual devem operar, a Internet, que de certa forma é considerado um ambiente hostil, traz consigo uma insegurança mesmo tendo em certos casos a necessidade de autenticação e de criptografia.

Portanto, os Serviços Web podem ser considerados uma grande evolução da Internet, mesmo tendo certas limitações, pois seus aplicativos podem ser construídos em qualquer tecnologia eliminando as limitações existentes entres as interfaces dos aplicativos, isto é, independência de linguagem e plataforma.

1.2 – *XML - Extended Markup Language*

A linguagem XML (*Extended Markup Language*) surgiu a partir de outras duas linguagens, a SGML (*Standard Generalized Markup Language*) e o HTML. A primeira versão foi disponibilizada pela W3C (*World Wide Web Consortium*) (W3C, 2002), com o objetivo de produzir um mecanismo simples e extensível para representação textual de informação estruturada e semi-estruturada. O XML possui a importante característica de ser extensível, permitindo que novos marcadores (*tags*) sejam criados por quem o utiliza (ABINADER, 2006).

Os marcadores são utilizados para identificar pedaços de informações ou nomes que são colocados entre sinais de menor (<) e maior (>), respectivamente. Há dois tipos de

marcadores, os iniciais e os finais; os iniciais são colocados entre o sinal menor (<) e o sinal maior (>), como em < inicio> e o final muito parecido com o inicial, porém após o sinal menor (<) existe a barra (/), como em </final>. Está noção de marcar inicio e final, possibilita o alinhamento permitindo que a informação seja estruturada hierarquicamente.

Para melhor entendimento da definição da linguagem XML, estão duas definições:

Segundo Faria (2005), descreve que XML é uma linguagem de marcação que consegue armazenar todo tipo de dado, descrevendo-os. Essa capacidade também chamada de *self-describe data* ou dados auto descritivos, por utilizar caracteres Unicode, pode armazenar qualquer tipo de caractere ou símbolo fazendo a representação de informações em qualquer idioma ou cultura.

Segundo Décio (2000), descreve que XML é uma linguagem usada para descrever e manipular documentos estruturados. A manipulação de documentos XML é feita por meio desta estrutura. Um documento XML é estruturado em forma de árvore possuindo sempre um elemento-raiz, de onde outros elementos vão se ramificar. Esta estrutura em árvore estabelece como os documentos XML vão ser definidos e como vão ser tratados. O tratamento envolve principalmente encontrar um determinado elemento ou um grupo de elementos para serem processados.

As aplicações Serviços Web utilizam o XML para transmitirem dados, assim fazem com que aplicações de desenvolvedores que aceitam o XML devem compreender qual a melhor forma de extrair informações a partir desse documento XML, para chegarem a seus resultados esperados.

Pela grande portabilidade da linguagem XML pode-se ter aplicações em diferentes plataformas se comunicando, basta que haja um protocolo para estabelecer esta comunicação e que o outro Serviço Web, feito em outra linguagem, entenda apenas a linguagem XML para se conectarem.

1.3 – *WSDL - Web Services Description Language*

A tecnologia WSDL (*Web Services Description Language*), que é feita em um documento XML, tem a função de descrever e localizar os Serviços Web de modo a informar ao usuário, o que o serviço executa, como o usuário pode invocar este serviço e como o usuário pode diferenciá-lo dos demais serviços, por serem oferecidos de diversos fornecedores de forma automática.

Segundo a W3C (W3C, 2002), WSDL é um documento em formato XML, que segue um *schema* XML próprio dos WSDL e é usado para descrever e localizar os Serviços Web. WSDL representa um contrato entre o cliente que requisita os serviços e o provedor de serviços que os oferece.

O usuário de Serviços Web precisa conhecer a assinatura do serviço, sua localização e o protocolo a ser usado para enviar a invocação; o documento WSDL fornece estas informações organizadas de duas maneiras lógicas: descrição abstrata e descrição concreta.

A descrição abstrata é composta por três elementos XML e por um sub-elemento do mesmo: o elemento *type,* contém tipos e definições independente de linguagem e plataforma; o elemento *message,* contém os parâmetros de entrada e saída e mensagens que já foram trocadas pelos serviços; o elemento *portTypes*, representa o conjunto de operações suportadas, métodos disponibilizados pelo serviço e mensagens envolvidas e o sub-elemento *operation*, representa a interação entre os serviços e as exceções permitidas durante as interações por meio das mensagens.

A descrição concreta é composta por dois elementos XML: o elemento *bindings*, especifica a ligação de cada operação do elemento *portTypes* e os associam com um protocolo de rede e o elemento *service*, é um conjunto de elementos *port* que descreve um endereço de rede para o elemento *binding.*

Todo documento WSDL possui um *definition* que é o elemento raiz do documento WSDL e, abaixo dele, são definidos todos os elementos que terão neste documento. Este elemento é o elemento mais externo do documento WSDL, ele contém o nome do Serviço Web e declarações relevantes de *namespaces.*

Além de todos estes elementos existem ainda mais dois elementos, o *import* e o *documentation.* O elemento *import* é usado para importar definições de WSDL de outros documentos WSDL e o elemento *documentation* é usado como comentário para leitores humanos entenderem aspectos do documento WSDL.

1.4 – *UDDI – Universal Description Discription and Integration*

O UDDI (*Universal Description Discovery and Integration*) foi criado para solucionar a rápida adoção do comércio via Internet. Ele é independente da W3C, diferentemente do SOAP e WSDL, e é mantido por empresas relacionadas a negócios (MCGOVER, 2003).

O seu objetivo é fornecer um modelo de dados padrão para armazenar informações sobre organizações e seus Serviços Web, por meio de um diretório que é independente de plataforma capaz de descrever, descobrir e integrar serviços usando a Internet (UDDI, 2007).

Clientes conseguem acessar o UDDI remotamente, para adicionar, atualizar, apagar e buscar informações no UDDI *registry,* pois define uma API baseada na comunicação via SOAP. O UDDI *registry* é uma base de dados padrão que suporta estruturas definidas pelo UDDI.

As estruturas de dados modelam informações sobre as organizações e os requerimentos técnicos para acessar os Serviços Web expostos por esta organização. O UDDI *registry*, pode ser pesquisado pelo nome da organização ou pelos Serviços Web disponibilizados pela organização.

A Figura 1.3 mostra como pode ser registrado a organização e seus Serviços Web no UDDI *registry*.

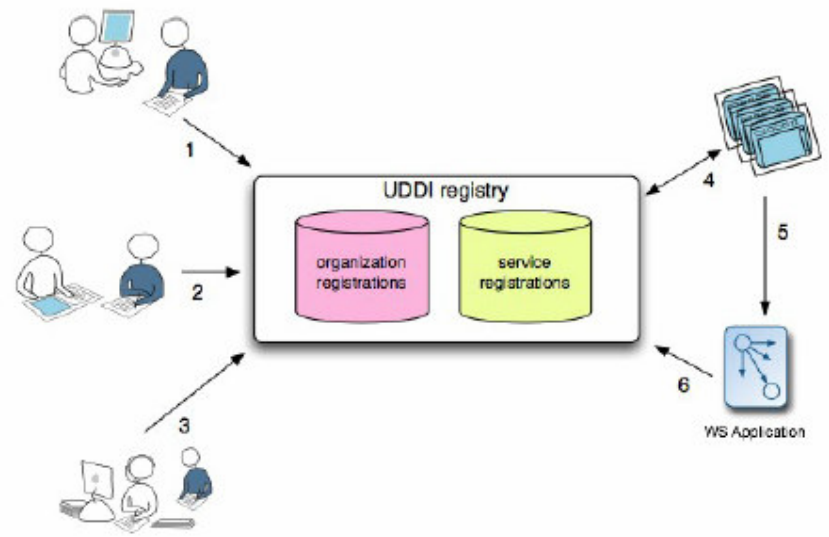

Figura 1.3 – Registro UDDI

Devido a grande flexibilidade oferecida pela linguagem XML e pelo elemento *import* presente na construção do WSDL, pode-se armazenar documento WSDL no próprio UDDI, que unidos, permitem grandes e complexas combinações entre as descrições de serviços, na qual são gerenciadas pela estrutura do UDDI (MCGOVER, 2003).

As dificuldades que a especificação UDDI pode auxiliar a resolver são: permitir encontrar a organização correta entre as milhões correntemente online; determinar como garantir trocas de informações uma vez que a organização correta foi descoberta; descobrir

novos clientes e aumentar o acesso aos atuais clientes; ampliar ofertas e se estender para outros mercados; remover barreiras para ingressar na economia globalizada pela Internet e descrever serviços e processos de negócios de maneira programática em um ambiente aberto e seguro.

1.5 – *SOAP – Single Object Acess Protocol*

Na arquitetura de Serviços Web, o protocolo padrão usado para comunicação entre duas máquinas é o SOAP (*Single Object Acess Protocol*). Neste protocolo, os serviços são acessados remotamente (RPC) por meio de mensagens formatadas que são desenvolvidas em formato XML e utilizam o protocolo HTTP para transportar os dados via Internet (MCGOVER, 2003).

"A grande inovação trazida pelo protocolo SOAP na computação distribuída, é a comunicação estabelecida entre sistemas distribuídos sobre uma coleção de softwares e hardwares heterogêneos" (MCGOVER, 2003).

A especificação do protocolo SOAP é formalizar como aplicações diferentes em ambientes distintos devem proceder para troca de mensagens. O protocolo SOAP tem a característica de escalabilidade, pois pode expandir para adequar a tecnologias e padrões já existentes e emergentes (MCGOVER, 2003).

Segundo Gomes (2002), o SOAP define uma estrutura simples para a realização de RPC's através da troca de documentos XML independentemente do protocolo. De acordo com Mendes (2002), a solicitação de um método é codificada usando o padrão SOAP e transmitida ao servidor usando o protocolo HTTP. O servidor, então, decodifica e decide qual ação tomar, que métodos serão chamados e que parâmetros serão considerados. Logo após a execução da ação requerida, o retorno também é codificado usando SOAP e enviado de volta ao cliente. O cliente recebe e decodifica a resposta e a passa como retorno da chamada ao método.

Resumindo, o SOAP é um protocolo de comunicação entre aplicações, um formato para enviar mensagens, projetado para comunicar via Internet, independente de plataforma e linguagem, baseado em XML, simples, extensível, permite passar mensagens por *firewalls*, utiliza o protocolo HTTP para transporte de dados e é um padrão da W3C.

A Figura 1.4 a seguir, ilustra uma pilha de comunicação de um Serviço Web.

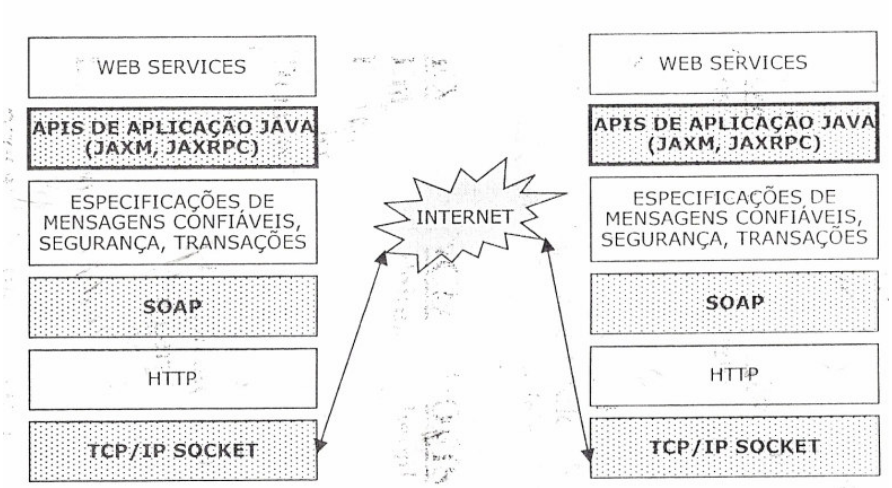

Figura 1.4 – Pilha de Comunicação de um Serviço Web. (ABINADER, LINS, 2006)

A estrutura do SOAP constitui de um documento no formato XML que contém o elemento raiz *Envelope*, o elemento *Header* (opcional) e um elemento indispensável, o *Body*, que pode conter anexos ou não (MCGOVER, 2003), conforme ilustrado na figura 1.5).

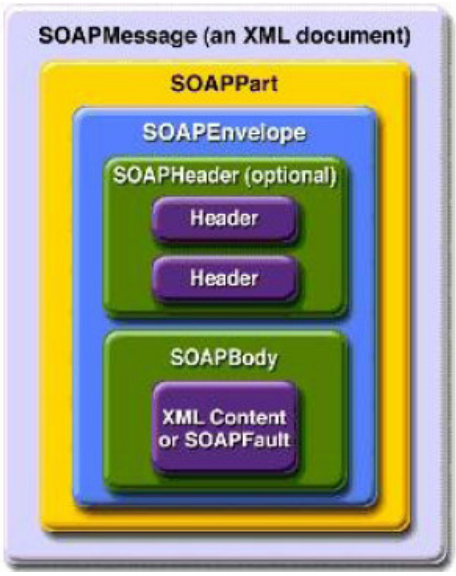

Figura 1.5 - Estrutura SOAP sem anexos

O elemento *Envelope* constitui-se da parte mais externa da estrutura SOAP, englobando os elementos *Header* e *Body*, ou seja, ele empacota as mensagens. Ele é responsável por indicar ao receptor onde começa e onde termina cada mensagem e se por acaso houver anexos, o marcador </Envelope> indica ao receptor que ele já os pode processar (MCGOVER, 2003).

O elemento *Header*, que é opcional numa mensagem SOAP e vem após o *Envelope*, processa novas instruções e contextos de informações para auxiliar as mensagens SOAP, como por exemplo, aumentar um conteúdo de uma mensagem, informar um novo roteamento, adicionar segurança, entre outros (MCGOVER, 2003).

O elemento *Body*, que é obrigatório numa mensagem SOAP, adiciona ao final da mensagem um conteúdo XML, que pode ser uma chamada a um procedimento definindo parâmetros e argumentos ou um documento, ou seja, ele finaliza a mensagem. Pode-se chamar estas duas possibilidades da seguinte forma respectivamente: mensagem SOAP estilo RPC e mensagem SOAP estilo documento (MCGOVERN, 2003).

O emissor troca mensagem com o receptor por meio das primitivas *send* e *receive*; no modo de mensagens *Request/Reply*, caso nesta troca de mensagem ocorra um erro, o protocolo SOAP define o elemento chamado *Fault* (subelemento de *Body*), que é responsável por manipular os erros contidos nas mensagens e os reportarem aos emissores dizendo qual foi o motivo pela não efetivação da troca de mensagem.

O elemento *Fault* se subdivide em dois elementos: *Code* e *Reason*.

· *Code*: precisa conter o subelemento *Value* com o valor do erro definido no *namespace* do SOAP. Os valores de *Value* podem ser:

· *VersionMismatch*: *namespace* do XML não bate com o do recipiente;

· *MustUnderstand*: atributo obrigatório do *header* não entendido;

· *DataEncodingUnknown*: o recipiente não suporta o *encoding* especificado;

· *Receiver*: ocorreu um erro no recipiente não relacionado com a mensagem;

· *Sender*: a mensagem não foi processada por erro do *sender* (falta informação ou mensagem mal formada).

· *Reason*: explicação do erro legível por humanos contida no sub elemento *Text*.

Os dois subelementos opcionais são:

· *Detail*: pode aparecer dentro dos outros subelementos e serve para conter mais detalhes do erro;

· *Node*: opcional quando vem de um *ultimate receiver* (ultimo *receiver*), obrigatório quando vem de um recipiente intermediário. Este elemento é inserido por uma aplicação antes do *ultimate receiver* que encontra um erro.

Existem quatro tipos de modelo de troca de mensagem: *Request/Reply*, *One-Way*, *Notification* e *Solicit/Response*, porém os mais utilizados são os dois primeiros.

O modelo *One-Way*, o usuário envia a mensagem para o Serviço Web, porém não espera uma resposta e no modelo *Request/Reply*, o usuário solicita a comunicação enviando

uma requisição e o Serviço Web a processa e envia uma mensagem “resposta”. Conforme ilustrado na Figura 1.7.

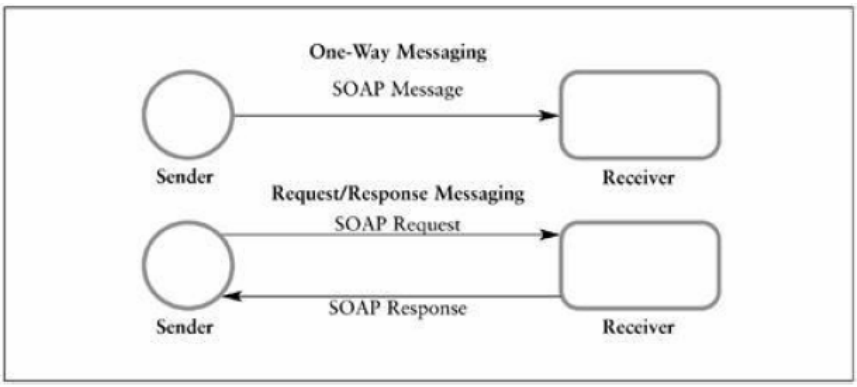

Figura 1.6 – Troca de mensagem *One-Way* e *Request/Reply*

1.6 – Aplicações de Serviços Web

Os Serviços Web possuem uma grande quantidade de aplicações na qual podem ser usadas e essa quantidade só tende a crescer por razão do aumento crescente da Internet e de aplicações multi-empresas, ou seja, as empresas não trabalham sozinhas, elas possuem fornecedores, parceiros e clientes.

Após a implantação de sistemas de informação na grande parte das empresas, notou-se a necessidade de integrar seus sistemas com os de outras empresas. O volume de dados vem aumentando rapidamente e isso vem provocando uma dificuldade cada vez maior para se alcançar à integração dos dados de forma eficiente e com qualidade. Neste ponto já é provável entender que os Serviços Web existem (principalmente) para a integração entre empresas (B2B).

Pode-se citar alguns exemplos dessas integrações:

- Integração de sistemas legados: na efetivação da integração de sistemas de terceiros, há a necessidade de disponibilizar equipes externas e internas, gastando assim um tempo e um custo por customizações. Empregando os Serviços Web, a própria equipe de desenvolvimento é capaz de programar a integração sem a necessidade de alteração no código dos sistemas em produção.
- Serviços Web: a própria equipe de desenvolvimento é capaz de programar a integração sem a necessidade de alteração no código dos sistemas em produção.
- Integração com entidades externas: com o uso de Serviços Web a integração com os fornecedores, clientes e parceiros fica mais rápida, mais eficiente, com maior

qualidade além de um baixo custo, porém se for levado em consideração o desempenho pelo fato da Web possuir “armadilhas”, esta solução não é indicada.

- Redução de custos operacionais: os Serviços Web proporcionam uma redução considerável nos custos, pois para a troca de dados utilizam a Web, enviam dados de maneira remota e possuem informações disponíveis on-line por meio da Web, extinguindo assim, a necessidade de se fazer estas operações fisicamente.
- Interfaces declaradas: duas equipes de desenvolvimento, por exemplo, não precisão se comunicar, seja por meio de conversa ou por meio de envio de documentos sobre o funcionamento de um projeto, pois apenas o documento de declaração de interfaces já fornece todos os dados necessários para realizar a conexão com o serviço, gerando assim, todo o código inicial do projeto.
- Novas oportunidades de lucro: no mercado há enormes possibilidades de lucros por meio de Serviços Web, basta ter experiência e criatividade. Um exemplo é uma cobrança por prestação de serviços remotos.

Aqui são citados alguns modelos de negócios bem sucedidos:

- Vantagem competitiva em relação a concorrentes;
- Diminuição de custos de operação, integração e manutenção;
- Redução no tempo da operação;
- Possibilidade de integração com plataformas heterogêneas;
- Agilidade nas transações;
- Venda de conteúdo jornalístico;
- Venda de indicadores de mercado;
- Venda de informações coletadas em tempo real;

Portanto, deve-se levar em consideração que os Serviços Web geram grandes benefícios, entretanto deve-se tomar cuidado com a deficiência do mercado, por adotarem padrões que às vezes torna difícil a escolha da tecnologia/plataforma a ser usada e até mesmo se existem profissionais capacitados para desenvolver e dar suporta em determinadas tecnologias.

CAPÍTULO 2 – TRABALHOS RELACIONADOS

Um trabalho que envolve Serviços Web com segurança foi feito por VARCHAVSKY (MUNDO JAVA, 2008). Esse autor ensina a desenvolver um simples Serviço Web acrescentando processos básicos de segurança usando o WSS4J por meio do CXF.

O serviço desenvolvido é uma classe Java que retorna uma String e possui um método principal que gera o serviço, atribuindo a uma variável o endereço do serviço; com a execução do mesmo, é gerado o WSDL e o serviço é publicado. Já o cliente foi desenvolvido em um projeto separado para mostrar a comunicação entre o servidor e o cliente apenas utilizando a troca de mensagem por meio do protocolo SOAP. É gerado um *Stub* no cliente por meio do utilitário WSDL2Java, para a comunicação do mesmo com o serviço.

Depois foi implementado um processo de autenticação por senha e configurações de criptografia. Toda a comunicação foi feita no padrão *WS-Security*, para facilitar a integração de segurança com diversos sistemas.

Outro trabalho que ensina a desenvolver Serviços Web por meio da ferramenta Axis foi desenvolvido por DESTRO (2008). Este trabalho explica desde as configurações básicas do Axis até os Jars necessários para o seu correto funcionamento. É demonstrada a criação de uma classe Java que executa lógicas relativas à área de recursos humanos.

Nesta classe Java é desenvolvido um método que inclui funcionário, um que conta o número de funcionários e um que calcula o seu salário em dólares, além do método construtor e dos métodos públicos *get()* e *set()*. Logo após é gerado o descritor WSDL por meio da ferramenta Java2WSDL do Apache Axis e, a partir deste descritor são criadas as classes chamadas *server-side* e o descritor do *deploy* (WSDD), por meio da ferramenta WSDL2Java, responsáveis pela organização do Serviço Web e pela invocação da classe de negócio.

Por fim é criada uma classe Java cliente responsável por consumir este Serviço Web. Nesta classe existe uma classe chamada *ServiceLocator*, que armazena e configura a URL para o Serviço Web e oferece método capaz de acessá-lo. Existe também uma classe *Binding*, responsável por realizar a comunicação entre o cliente e o Serviço Web, traduzindo a comunicação entre o Java e o Serviço Web. Após isto é invocado o serviço desejado e passado os devidos parâmetros para a execução dos mesmos.

KABUTS (2005) também apresenta um tutorial desenvolvido com a ferramenta Apache Axis e mostra o desenvolvimento de um servidor, um cliente para o mesmo e um cliente que acessa um serviço disponibilizado pela *Google*.

Na parte servidor é desenvolvido uma classe Java que calcula o *Fibonacci*, é colocada dentro do diretório do Axis e renomeado com a extensão jws (*Java Web Service*). A partir de então, é acessado o serviço por meio de uma URL e o descritor a partir de um link da URL, descrevendo a interface da classe por meio de uma linguagem universal (XML).

O cliente desenvolvido para consumir este Serviço Web é uma classe Java simples, que depois de adicionado o utilitário WSDL2Java ao Eclipse, basta criar um serviço do tipo *FibonacciService*, instanciar um *FibonacciServiceLocator* e invocar o método desejado no Serviço Web passando como parâmetro o número desejado para calcular o F*ibonacci*. No caso deste tutorial é passado como parâmetro o número dentro de um comando *for,* para que seja calculado o *Fibonacci* de n números dependendo da condição do comando.

Já a *Google* publicou uma interface para um Serviço Web que disponibiliza um serviço de busca e o seu WSDL está disponível em http://api.google.com/GoogleSearchwsdl. A empresa *Google* tornou este arquivo disponível para o público em geral, porém existem algumas restrições, como por exemplo, um limite de solicitações a este serviço e usuários que querem acessar esse tipo de serviço, precisam ser cadastrados no site para obter uma licença de uso.

Para desenvolver o cliente que utilize este serviço, primeiro foi necessário gerar classes Java do WSDL do *Google* a partir do utilitário WSDL2Java. O autor utiliza as classes geradas para efetuar uma pesquisa por nomes famosos e as organiza por estimativas retornadas do Serviço do *Google* para o cliente informando quem é mais famoso.

O cliente deste Serviço Web é basicamente igual ao anterior (contém o *GoogleService e o GoogleServiceLocator*), porém é definido uma porta para o *Google* e uma chave que é a licença adquirida no site do mesmo. As pessoas pesquisadas são atribuídas a uma variável do tipo *String,* processadas dentro de um comando *for* que faz as requisições dos nomes ao servidor, comparadas dentro de outro método que as coloca em ordem crescente e por fim são impressas na tela.

Um outro tutorial disponível na Internet (MARILIA, 2008), desenvolve um cliente e um servidor de Serviço Web em Java usando Eclipse, Axis2 1.4 e Tomcat 6. Neste tutorial são mostradas as configurações do Eclipse para que suas aplicações utilizem o *plugin* do Axis2, sem haver a necessidade de ter que transformar esta classe em um arquivo jws e coloca-la dentro do diretório do Axis. O serviço oferecido é uma classe Java que disponibiliza

dois métodos, um que faz conversão de real para *dolar* e outro que faz de *dolar* para real, que com o *plugin* Axis2, este transforma os métodos da classe Java em Serviços Web e os publica.

Já o projeto do cliente desta aplicação, que também é feito utilizando o *plugin* do Axis2 instalado no Eclipse, já é passado a URL que ele deve acessar para utilização do serviço durante a sua configuração.

Com o projeto do cliente já criado, agora é necessário criar uma classe Java dentro do mesmo para utilização dos Serviços. A classe Java é bem simples, é apenas criada uma variável do tipo *Stub* passando a URL do Serviço Web desejado como parâmetro, depois é invocado o objeto *request* para a solicitação do método requerido, em seguida é passado o valor por meio do objeto *request* ao Serviço Web e por fim é invocado o objeto *response* do mesmo, para receber o novo valor adquirido após a execução do Serviço Web.

Todos estes trabalhos relacionados acima foram desenvolvidos de maneira superficial, ou seja, o usuário ou engenheiro de software que tem interesse em desenvolvê-los, tem que possuir certo nível de conhecimento do assunto, pois o tutorial demonstra que certos passos já são do conhecimento de quem o está desenvolvendo.

O próximo capítulo apresenta o desenvolvimento de um tutorial didático e detalhado, levando em consideração que os mesmos possam ser leigos no assunto.

CAPITULO 3 - TUTORIAL SERVIÇO WEB

3.1 – Projeto Proposto

Este trabalho tem como objetivo apresentar um tutorial que possa ser utilizado por engenheiros de software para o desenvolvimento de aplicações simples que utilizam serviços Web. O tutorial é apresentado neste capítulo e utiliza como exemplo um sistema hipotético de uma agenda. Descreve-se passo a passo o desenvolvimento desse sistema de forma que pessoas interessadas possam replicar este conhecimento em seus ambientes de trabalho ou acadêmicos. Os Serviços Web implementados são inserção, remoção e busca de contatos.

No projeto, foram desenvolvidas as duas partes que compõem a arquitetura de um Serviço Web – cliente e servidor. O Servidor tem como responsabilidade disponibilizar estes serviços citados acima, enquanto que o lado cliente tem como objetivo utilizar estes serviços para uso próprio.

As tecnologias utilizadas para o desenvolvimento do sistema apresentado neste tutorial foram o Apache Axis; um *framework* para se construir descritores WSDL e se comunicarem por meio do protocolo SOAP e o Apache Tomcat, um container web. Por meio do Axis, foi possível desenvolver os Serviços Web e o cliente desse serviço, pelo fato do Axis ser facilmente integrado à sua aplicação *web*, independentemente do *container*.

Foram utilizados outros softwares para dar suporte ao desenvolvimento, como o banco de dados *MYSQL* e o *Net Beans*, um ambiente para o desenvolvimento Java. O banco de dados *MySQL* é simples e faz o serviço de armazenamento, já o ambiente de desenvolvimento *Net Beans 6.0.1*, é uma ferramenta Java que possui a maior parte dos requisitos necessários para auxiliar no desenvolvimento do Serviço Web em Java. Uma grande vantagem de desenvolvê-los nestas ferramentas está pelo fato de todas serem *Open Source*, ou seja, softwares livres.

Existem inúmeras ferramentas disponíveis para desenvolvimento de Serviço Web, alguns exemplos de *frameworks* são: Axis, *XFire*, JAX-WS (*Java API for XML Web Services*), REST (*Representational StateTransfer*), dentre outras.

Por fim é criado um JAR de toda parte de banco de dados e persistência, colocando-o dentro da pasta *lib* de cada ferramenta para que esta o utilize. Esta operação é descrita na seção 3.3.

3.2 – Preparação das Funcionalidades da Agenda

Para iniciar o desenvolvimento deste Serviço Web, primeiramente é necessário ativar o banco de dados e criar uma tabela que contem os seguintes campos: id do tipo inteiro, chave primária e *not null* e nome, telefone e endereço do tipo *varchar*, que são responsáveis por armazenar os dados do usuário no banco de dados, conforme ilustrado na Figura abaixo.

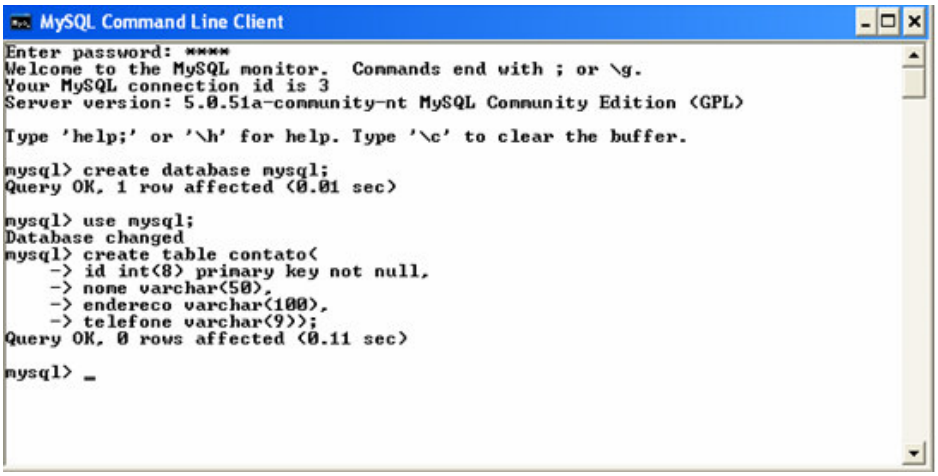

Figura 3.1 – Criação da tabela contato

Com a tabela já criada, é necessário programar toda a parte de conexão, inserção, remoção e busca do ambiente Java com o banco de dados, que são mostrados nas páginas subsequentes. Depois de implementados são transformados em uma biblioteca (.jar) e colocados dentro da pasta *lib* do Axis, para que as aplicações os utilizem de maneira a disponibilizar os serviços no Servidor, isto é, essas três classes são as três operações disponíveis.

Inicialmente é necessário desenvolver uma classe Java chamada Contato que possui alguns atributos convencionais e tipos primitivos da linguagem Java. Os métodos *sets* e *gets* e o construtor, embora não mostrados, devem ser criados pelo desenvolvedor, conforme ilustrado no apêndice A.

Os métodos públicos *get ()* e *set ()* são responsáveis por fazerem o acesso externo de atributos de classes privadas (*get( )* retorna o atributo e o *set( )* o seta/modifica), pelo fato de um atributo privado só ser acessado internamente pela própria classe, há a necessidade de instancia-los para tal acesso(DEITEL, 2003, pag.391).

Após a elaboração deste método, é necessário construir o método responsável por estabelecer a conexão com o banco de dados, conforme ilustrado na Figura 3.2.

```
       public class Conexao {
          //Método que estabelece a conexão com o banco de dados
           public static Connection getConnection() throws
SQLException {
               try {
                   Class.forName("com.mysql.jdbc.Driver");
                   Return DriverManager.getConnection
("jdbc:mysql://" + "localhost/mysql", "root", "root");
               } catch (ClassNotFoundException e) {
             System.out.println("Não foi possivel encontrar o
 driverde  " + " banco: " + e.getMessage());
                   throw new SQLException(e.getMessage());
               }
           }
       }
```

Figura 3.2 – Conexao.java

Na Figura 3.3 é muito importante ressaltar que toda conexão com o banco de dados tem de possuir obrigatoriamente um *try/catch* (tratamento de exceção) e na parte do *return DriverManager. getConnection("jdbc:mysql://" + "localhost/mysql", "root", "root"),* o *mysql* é o nome do banco que foi defino no *prompt* do MYSQL, *localhost/mysql* quer dizer que o banco é local e *root / root* são respectivamente usuário e senha que foram definidos na instalação do MYSQL.

Com a parte da conexão já feita, agora é necessário criar a classe de persistência, responsável por inserir, remover e buscar informação no banco de dados. Na Figura 3.3 a seguir, pode-se observar uma parte desta classe de persistência sendo que o restante encontra-se no apêndice C.

```
            //Método que insere no Banco
           public void inserirContato(Contato contato) throws
SQLException {
                PreparedStatement pre =
                this.connection.prepareStatement ("insert" + " into
                contato(id, nome, telefone, endereco) values(?, ?, ?,
                ?)");
                pre.setInt(1, contato.getId());
                pre.setString(2, contato.getNome());
                pre.setString(3, contato.getTelefone());
                pre.setString(4, contato.getEndereco());
                pre.execute();
                pre.close();
                connection.close();
                System.out.println("GRAVADO");
            }
            public Contato buscarNome(String nomeContato) throws
SQLException {
                PreparedStatement pre =
                this.connection.prepareStatement ("select * " + "from
                contato where nome = ?");
                pre.setString(1, nomeContato);
                Contato newContato = new Contato();
                ResultSet rSet = pre.executeQuery();
                if (rSet.next()) {
                    newContato.setNome(rSet.getString("nome"));

newContato.setEndereco(rSet.getString("endereco"));

newContato.setTelefone(rSet.getString("telefone"));
                    return newContato;
                } else {
                    return null;
                }//se não achar vai retornar null
            }
            // Método que deleta do Banco
           public void DeletarContato(Contato contato) throws
SQLException {
                PreparedStatement pre =
this.connection.prepareStatement (            "delete" + " from
contato where nome = ?");
                pre.setString(1, contato.getNome());
                pre.execute();
                pre.close();
                connection.close();
                System.out.println("DELETADO");}
            }
```

Figura 3.3 – ContatoDAO.java

Os valores da inserção são passados como “? ? ? ?” , pois o que é gravado no banco é o que for passado por parâmetro na aplicação do cliente. Depois chamam os métodos *pre.setInt* e *pre.setString*, passando como parâmetro a ordem que são armazenados e o que é armazenado. Agora é só chamar o *pre.execute()* para executar o *PreparedStatement* e depois

fecha-lo por meio dos método *pre.close()* e fechar a conexão através do método *connection.close()*, conforme ilustrado na Figura 3.3.

Já o método buscarContato(), cria-se uma variável *pre* do tipo *PreparedStatement* responsável por preparar o banco de dados para a busca. Na condição da busca (*select*), é passado o valor "?", pois o que é buscado no banco de dados é o que for passado por parâmetro na aplicação do cliente, conforme ilustrado na Figura 3.3.

Logo em seguida chama-se o método *pre.setString* para setar o nome que foi passado por parâmetro, cria-se uma variável chamada *newContato* do tipo Contato e uma *rSet* do tipo *ResultSet*, recebendo o método *pre.executeQuery*.

É criada a variável rSet do tipo ResultSet, pois é ela que vai percorrer todo o banco de dados à procura do nome a ser buscado e é o método pre.executeQuery que fará o inicio desta busca.

A condição *if(rSet.next)* é responsável por percorrer o banco de dados, ou seja, enquanto existir um próximo contato, ele irá comparar o nome, o endereço e o telefone da variável *newContato*, pelo contato em que o rSet esta armazenado temporariamente, caso seja o que está sendo procurado, retorna o contato, caso contrário, retorna NULL, conforme ilustrado na Figura 3.3.

Por fim, o método *DeletarContato()* tem como parâmetro o contato a ser deletado e nele é criado uma variável *pre* do tipo *PreparedStatement*, responsável por preparar o banco de dados para deletar. Em seguida chama-se o *pre.setString()* passando como parâmetro dois argumentos, a posição do banco de dados e o que ele seta, que neste caso é o nome a ser deletado. Após isto invoca o método *pre.execute* para execução do mesmo e por fim fecha-se o *PreparedStatement* e a conexão por meio dos métodos, *pre.close* e *connection.close* respectivamente, conforme ilustrado na Figura 3.3.

Após a implementação destes métodos é necessários compilar todas estas classes e colocá-las no diretório WEB-INF\classes do Axis ou empacotar todas elas em um JAR e colocá-las no diretório WEB-INF\lib do mesmo. Neste caso foram transformadas em JAR pelo Net Beans da seguinte maneira: no projeto em que foram implementados as operações, clique com o botão direito e depois em construir, conforme ilustrado na Figura 3.4. Logo após é criado o JAR no diretório padrão do Net Beans, que se localiza em: Meus documentos\NetBeansProjects\BancoDeDados\dist, sendo BancoDeDados o nome do projeto na qual foi desenvolvido as classe e dist a pasta onde se encontra o novo JAR criado. No capítulo a seguir é mostrado como colocar este JAR na pasta do Axis.

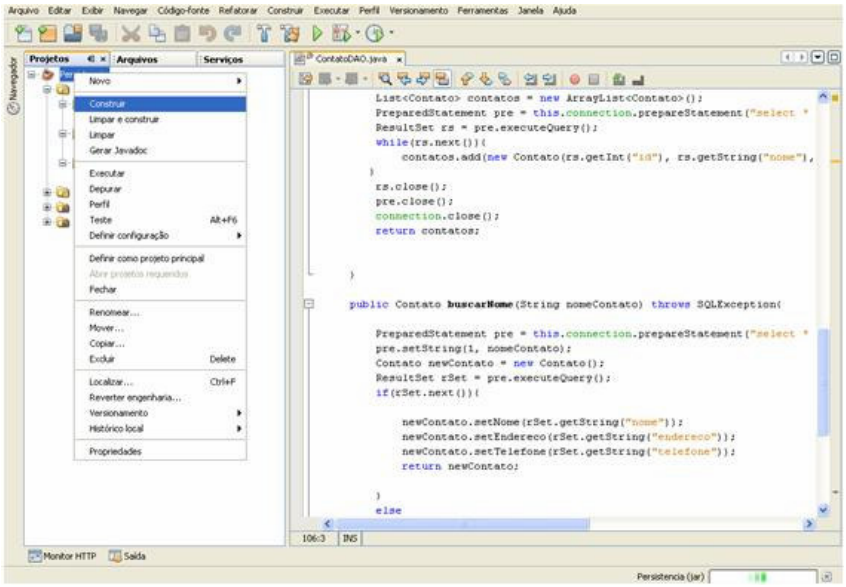

Figura 3.4 – Criando o JAR

3.3 – Preparação do Ambiente

Inicialmente é necessário instalar o *web container Tomcat*, que pode ser encontrado no site da Apache (http://tomcat.apache.org/download-60.cgi#6.0.18). Após o seu *download*, instale-o no *Hard Disk* (Disco Rígido), o inicie, logo em seguida abra o *browser* e digite o seguinte endereço: http://localhost:8080, conforme ilustrado na Figura 3.5.

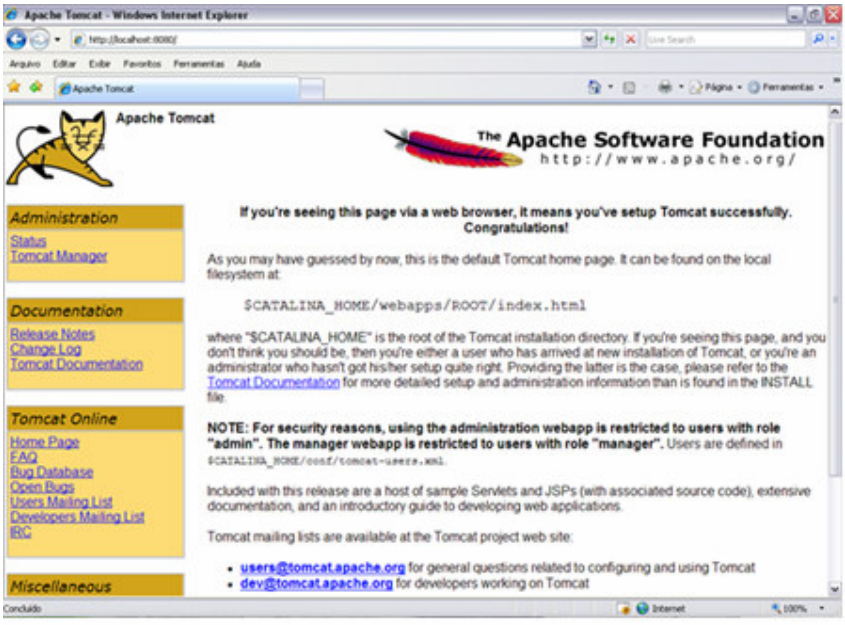

Figura 3.5 – Tomcat

Caso o Tomcat execute corretamente, abre a tela ilustrada na Figura 3.5, caso contrário, abre uma tela dizendo os motíveis prováveis pela não execução do mesmo.

O *download* do Axis é feito pelo site da apache (http://www.apache.org/dyn/closer.cgi/ws/axis/1_4/), depois é necessário descompactar o *zip* do mesmo no disco rígido e colocar a pasta do axis, localizado em AXIS_HOME\webapps\ ,dentro do diretório webapps do Tomcat, que por padrão se localiza em: C:\Arquivos de programas\ Apache Software Foundation\Tomcat 6.0\webapps.

Com o Axis instalado no Tomcat é necessário fazer o *download* de alguns arquivos JARs e coloca-los dentro da pasta lib do Axis, que se encontra em: C:\Arquivos de programas\ Apache Software Foundation\Tomcat 6.0\webapps\axis\WEB-INF\lib, conforme ilustrado na Figura 3.6.

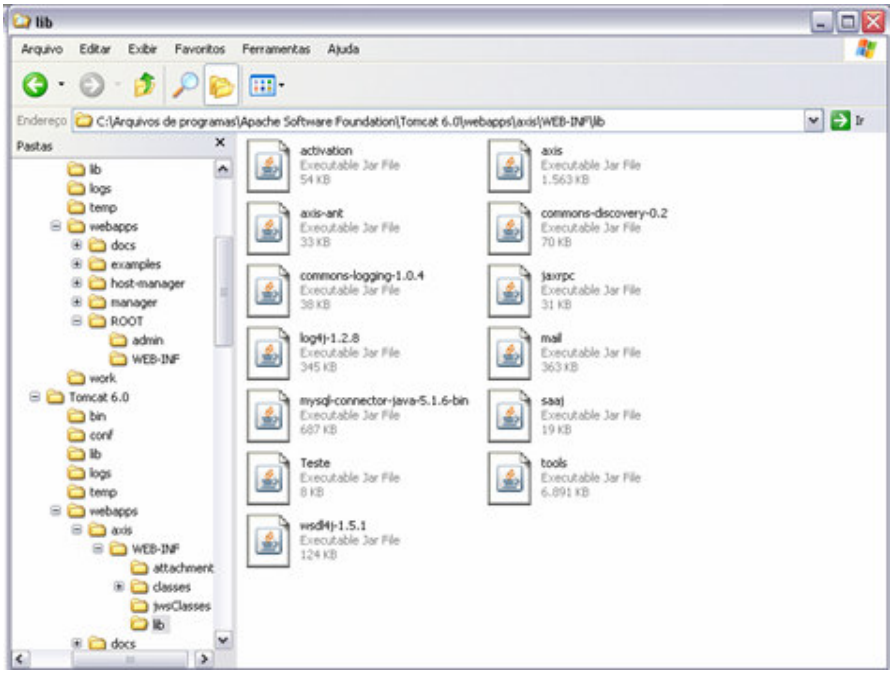

Figura 3.6 – Diretória lib do Axis

Neste diretório *lib* se encontrada todos os JARs necessários para o funcionamento do Axis, como por exemplo, a conexão com o banco de dados (*mysql-connector-java-5.1.6-bin*), o JAR da persistência criado na seção 3.2 , o arquivo wsdl4j-1.5.1, necessário para o Axis interpretar o XML do WSDL, dentre outros.

Para testar o funcionamento do Axis basta digitar na barra de endereço de algum *browser* o seguinte endereço: http://localhost:8080/axis, conforme ilustrado na Figura 3.7.

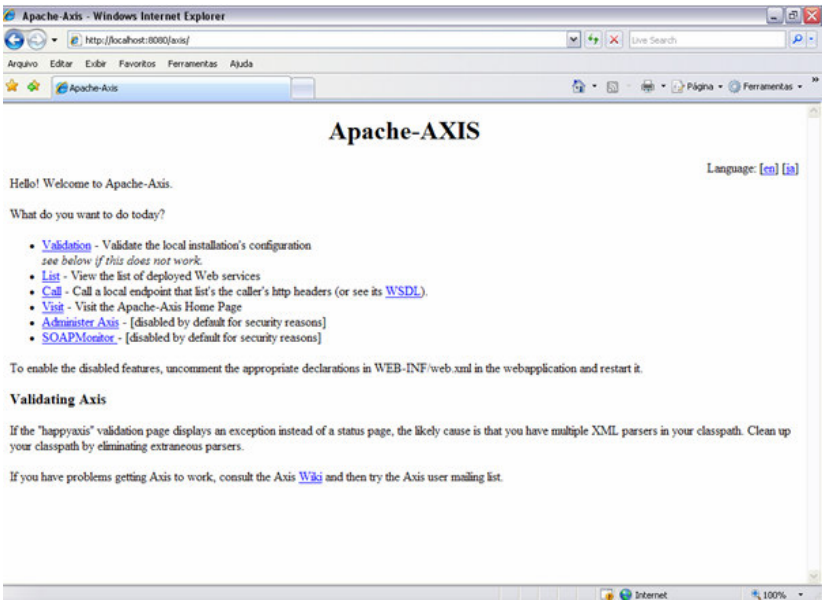

Figura 3.7 – Apache Axis

Clicando-se no primeiro *link* disponível da página (*Validate),* deve-se ver um resumo geral do Axis, conforme ilustrado na Figura 3.8. Caso ocorra algum tipo de erro (como falta de algum JAR), deve ser corrigido antes de prosseguir e também deve ser observado que as *core axis libraries* devem ser encontradas (a biblioteca do Axis). Caso falte algum JAR, o próprio Axis vai lhe informar o site disponível para o *download* do mesmo, que depois de feito, o mesmo é colocado na pasta lib do Axis. Por fim feche o Tomcat e faça todos os procedimentos do início desta seção para a validação do novo JAR adicionado.

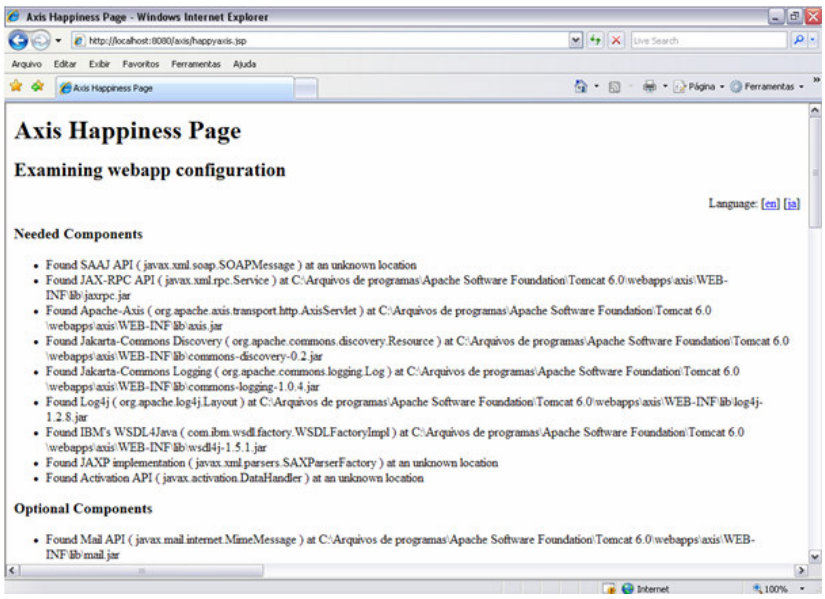

Figura 3.8 – Validate Apache Axis

Com todos estes passos descritos ao longo deste capítulo, o ambiente onde está sendo desenvolvido o Serviço Web já está apto para funcionar corretamente e, a partir de então, é descrito o desenvolvimento dos serviços oferecidos, dos clientes e como os mesmos funcionam.

3.4 – O Servidor

O servidor é aquele responsável por disponibilizar os serviços para os demais clientes, a seguir é mostrado o código responsável pelo evento.

```
public class Servico {
       public Object[] criarContato(int id, String nome, String
       telefone, String endereco) {
            Contato contato = new Contato(id, nome, telefone, endereco);
                  try {
                        ContatoDAO contatoDao = new ContatoDAO();
                        contatoDao.inserirContato(contato);
                  } catch (SQLException e) {
                   return null;}
                  List<Object> contatoList = new ArrayList<Object>();
                  contatoList.add(id);
                  contatoList.add(nome);
                  contatoList.add(telefone);
                  contatoList.add(endereco);
                  return contatoList.toArray();}
            public Object[] buscarContato(String nome) {
                  try{
                     ContatoDAO contatoDAO = new ContatoDAO();
                     Contato contato = contatoDAO.buscarNome(nome);
                     if (contato != null){
                        List<Object> contatoList = new ArrayList<Object>();
                              contatoList.add(contato.getId());
                              contatoList.add(contato.getNome());
                              contatoList.add(contato.getTelefone());
                              contatoList.add(contato.getEndereco());
                              return contatoList.toArray();
                        } else
                              return null;
                  } catch (SQLException e) {
                        return null;}}
            public boolean deletarContato ( String nome ) {
                        try {
                        ContatoDAO contatoDao = new ContatoDAO();
                        Contato contato = contatoDao.buscarNome(nome);
                        if (contato!= null) {
                               contatoDao.DeletarContato(contato);
                               return true;}
                         else
                          return false;
                        } catch (SQLException e) {}
                  return true;}}
```

Listagem 3.9 – Servico.java

Como pode ser notado esta classe chamada Servico é responsável por disponibilizar toda a parte do serviço que é prestada aos clientes (inserção, remoção e busca), no entanto, para desenvolver o descritor (WSDL) é necessário salvar esta classe Java no formato jws (maneira na qual o Axis interpreta gerando todo o WSDL) e colocá-la no diretório do axis localizado em: C:\Arquivos de programas\Apache Software Foundation\Tomcat 6.0\webapps \axis.

O Servidor já está pronto, agora basta entrar no Axis e mandá-lo gerar todo o código XML por meio do endereço: http://localhost: 8080/axis/Servico.jws, onde Servico.jws é o nome do Serviço Web que disponibiliza as operações. Para visualizar o descritor WSDL do mesmo é necessário clicar no link da mesma página chamado, *Click to see the WSDL* (Clique para ver o WSDL), conforme ilustrado na Figura 3.10.

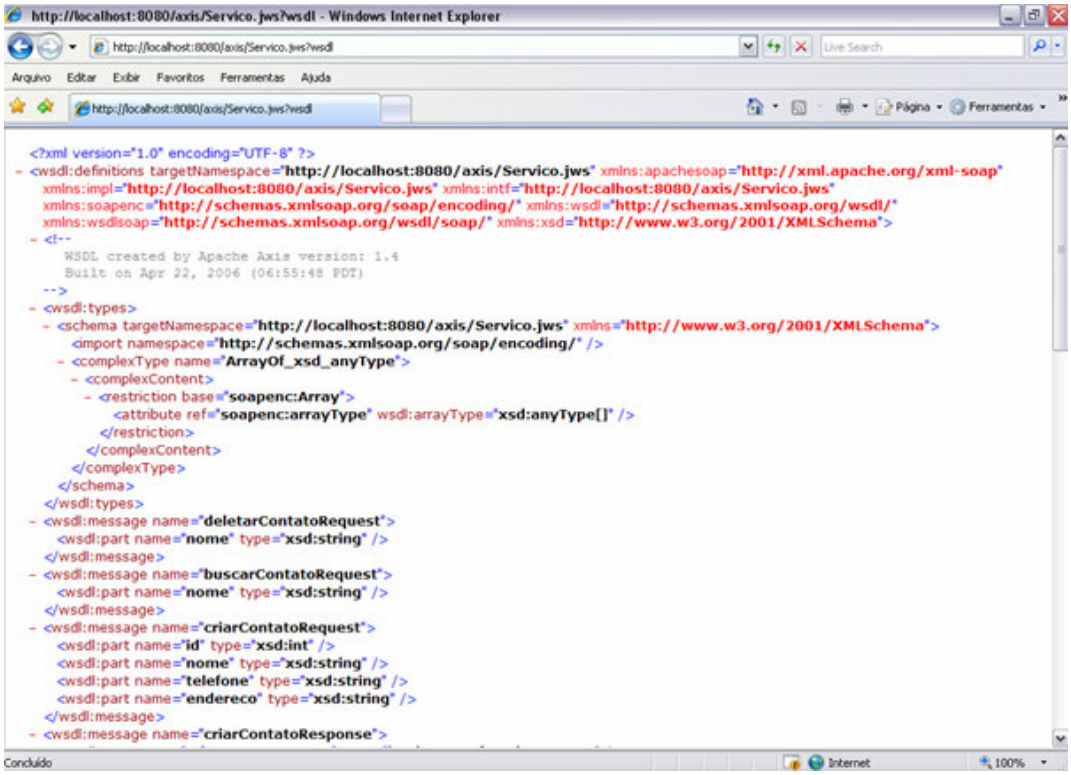

Figura 3.10 – WSDL Servidor

Caso queira criar outro tipo de serviço, faça uma classe Java que disponibiliza os mesmos e repita todos os procedimentos desta seção 3.4.

Na Figura 3.11 a seguir, mostra-se uma parte do descritor do servidor gerado pelo Axis a partir da classe Serviço.jws, sendo que o restante encontra-se no apêndice E.

```
?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace=http://localhost:8080/axis/Servico.jws
<wsdl:types>
<schema targetNamespace=http://localhost:8080/axis/Servico.jws
 xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
 <complexType name="ArrayOf_xsd_anyType">
 <complexContent>
 <restriction base="soapenc:Array">
<attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:anyType[]" />
</wsdl:message>
 wsdl:message name="buscarContatoRequest">
<wsdl:part name="nome" type="xsd:string" />
</wsdl:message>
</wsdl:message>
<wsdl:message name="buscarContatoResponse">
< wsdl:part name="buscarContatoReturn" type="impl:ArrayOf_xsd_anyType" />
</wsdl:message>
 <wsdl:portType name="Servico">
</wsdl:operation>
 <wsdl:operation name="buscarContato" parameterOrder="nome">
<wsdl:input message="impl:buscarContatoRequest" name="buscarContatoRequest" />
<wsdl:output message="impl:buscarContatoResponse" name="buscarContatoResponse"/>
</wsdl:operation>
</wsdl:portType>
   <wsdl:binding name="ServicoSoapBinding" type="impl:Servico">
 <wsdlsoap:binding style="rpc"transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="buscarContato">
<wsdlsoap:operation soapAction="" />
 <wsdl:input name="buscarContatoRequest">
<wsdlsoap:body encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
  namespace="http://DefaultNamespace" use="encoded" />
</wsdl:input>
<wsdl:output name="buscarContatoResponse">
 <wsdlsoap:body encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
   namespace="http://localhost:8080/axis/Servico.jws" use="encoded" />
 </wsdl:output>
 </wsdl:operation>
< wsdl:service name="ServicoService">
  wsdl:port binding="impl:ServicoSoapBinding" name="Servico">
  wsdlsoap:address location="http://localhost:8080/axis/Servico.jws" />
 </wsdl:port>
 </wsdl:service>
 </wsdl:definitions>
```

Listagem 3.11 – Código WSDL do Servidor

A partir deste código XML mostrado acima, todo cliente em qualquer plataforma e linguagem é capaz de usufruir deste serviço, basta entender a linguagem XML e o protocolo SOAP. Na linguagem XML acima é informado ao cliente tudo o que ele precisa saber sobre o

serviço, como por exemplo, o endereço onde se localiza, quais funções podem ser utilizadas, quais parâmetros são necessários para cada função e qual protocolo é utilizado.

Em *"<schema targetNamespace=" http://localhost:8080/axis/Servico.jws "">*, o *schema* especifica que o vocabulário criado é válido em um namespace, que neste caso, usou-se uma URL (*Uniform Resource Locator*) para identificar o namespace, lugar onde todos os nomes são únicos e onde são localizados pelos clientes.

O *"<wsdl:operation name="buscarContato" parameterOrder="nome">"*, mostra o nome da operação disponível e indica quais são os parâmetro de entrada necessários com suas respectivas ordens.

O *"<wsdl:message name="buscarContatoRequest">"* e o *"<wsdl:part name="nome" type="xsd:string" />"*, definem que o método invocado tem este específico parâmetro (nome) do tipo *String*.

O *"<wsdl:message name="buscarContatoResponse">"* e o *"<wsdl:part name="buscarContatoReturn" type="impl:ArrayOf_xsd_anyType" />"*, definem que este método retorna um *array* de algum tipo que foi definido anteriormente (<complexType name="ArrayOf_xsd_anyType">).

Por fim é descrito qual o nome do Serviço Web e o caminho que o localiza (URL), para que seja acessado por meio do protocolo SOAP remotamente *("<wsdl:service name="ServicoService">"*, *"<wsdl:port binding="impl:ServicoSoapBinding" name="Servico">"* e *"<wsdlsoap:address location="http://localhost:8080/axis/Servico.jws" />"*).

As demais operações descritas neste arquivo XML (criarContato(id,nome,telefone, endereço) e deletarContato(nome)), seguem o mesmo projeto de elaboração deste, conforme apontado no apêndice E.

Para que este Serviço Web seja localizado e encontrado por uma aplicação é necessário que o registre no site da UDDI, na qual pode ser acessado remotamente e, a partir de então, quem tiver interesse em usá-lo, acesse o site, procure pelo nome que o mesmo foi registrado, obtendo assim, o endereço na qual a aplicação irá referência-lo.

3.5 – O Cliente

O cliente utiliza os serviços disponibilizados pelo Servidor do Serviço Web, podendo ser uma aplicação que a referencia para utilização dos mesmos ou simplesmente um cliente específico para a mesma utilização. Neste caso o endereço do Serviço Web já está visível para

a sua utilização (“http://localhost:8080/axis/ Servico.jws”) e portanto, os três clientes desenvolvidos são específicos para esta utilização e já podem referenciá-lo a partir desde endereço.

Primeiramente é necessário testar se o serviço está funcionando corretamente, para isso, abra o *browser* e na barra de endereço digite: http://localhost:8080/axis/ Servico.jws? method=criarContato&id=1&nome=Anthony&telefone=34336849&endereco=Francisco, conforme ilustrado na Figura 3.12.

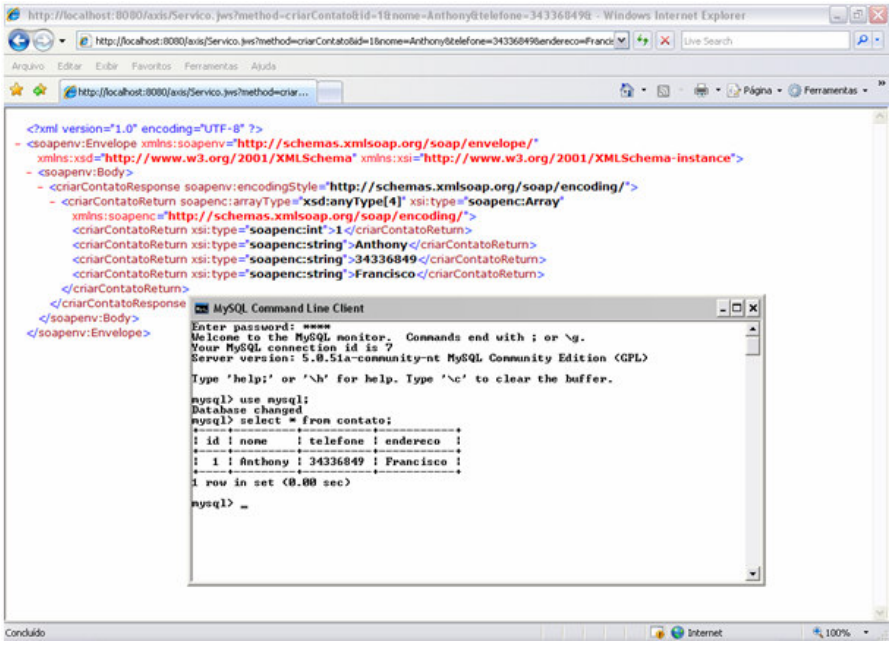

Figura 3.12 – Teste Cliente-Servidor

Pode-se notar que ao executar o endereço com o método criarContato e os parâmetros id, nome, telefone e endereço, o Serviço Web retorna o que é inserido no banco para que exista um *feedback* do que aconteceu, caso dê errado, retorna o tipo do erro.

Como o funcionamento correto deste Serviço Web, a seguir é mostrado o código do cliente responsável por utilizar o serviço de inserção na agenda. Vale ressaltar que tipos como coleções são recebidos como *arrays* de objetos, por que nem todas as linguagens de programação possuem coleções (JANDL, 2002).

```
public class Cliente_Inserção {
      public static void main(String[] args) throws Exception {
        // Endereço, local onde se encontra o Web Service
         String local = "http://localhost:8080/axis/Servico.jws";
        // Criando e configurando o serviço
         Call call = (Call) new Service().createCall();
        // Configurando o endereço.
         call.setTargetEndpointAddress(local);
        // Marcando o método a ser chamado.
         call.setOperationName("criarContato");
        // Parâmetros da função criarContato.
         Object[] param = new Object[]{1, "Anthony", "3481-6924",
"Francisco 828"};
        // Retorno da Função
         Object[] ret = (Object[])call.invoke(param);
        // Imprime o resultado
         System.out.println(" Inserido com Sucesso " );
      }
}
```

Figura 3.13 – Cliente_Inserção.java

Na Figura 3.13 é criada uma variável chamada local do tipo *String* e uma *call* do tipo *Call*, responsáveis por armazenar o endereço onde se localiza o Serviço Web e por criar e configurar a chamada ao serviço, respectivamente.

O comando "*call.setTargetEndpointAddress(local)*", configura o endereço armazenado na variável local e o comando *"call.set.OperationName("criarContato"),* marca qual o método é invocado.

A seguir cria-se uma variável chamada "param", um *array* do tipo *Object,* e instancia-a passando como parâmetros id, nome, telefone e endereço, que são os parâmetros definidos no método que é invocado (criarContato).

Por fim cria-se uma variável chamada "ret" do tipo *array* de *Object* que recebe o método "*call.invoke(param)"*, responsável por fazer o retorno do método criarContato. Logo chama-se o método "*System.out.println()"* passando como parâmetro o que é escrito na tela, que neste caso é: "Inserido com Sucesso".

Quando este cliente é executado ele busca o serviço criarContato na interface WSDL e apresenta o seguinte resultado.

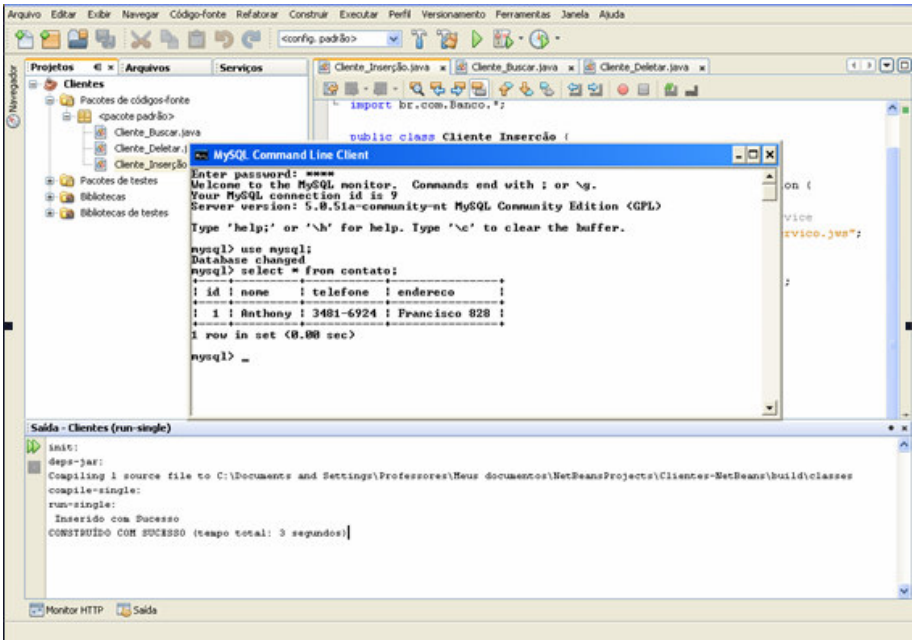

Figura 3.14 – Resultado Inserção

Pode-se observar que quando acionado o serviço, ele grava no banco o que lhe foi passado por parâmetro e lhe retorna uma mensagem que é transformada no retorno do método invocado, que neste caso é a impressão na tela que a inserção foi realizada com sucesso, conforme ilustrado na Figura 3.14.

Os métodos buscarContato() e deletarContato(), que também estão disponíveis aos clientes, seguem o mesmo modelo do código da Figura 3.13, porém se diferem nos parâmetro e no tipo de retorno, conforme mostrado nas Figuras 3.15.

```
    public class Cliente_Buscar {
     public static void main(String[] args) throws Exception {
       // Endereço, local onde se encontra o Web Service
           String local = "http://localhost:8080/axis/Servico.jws";
          // Criando e configurando o serviço
           Call call = (Call) new Service().createCall();
       // Configurando o endereço.
           call.setTargetEndpointAddress(local);
       // Marcando o método a ser chamado.
           call.setOperationName("buscarContato");
       //Parâmetros da Funçãoo
           Object[] param = new Object[]{"Anthony"};
       //Retorno da Função
           Object[] ret = (Object[])call.invoke(param);
       //Imprimindo o Resultado
        System.out.println("ID: " + ret[0] + "\nNome: " + ret[1] +
"\nTelefone: " + ret[2]+ "\nEndereÃ§o: " + ret[3]);
       }
     }
```

Figura 3.15 – Cliente_Buscar.java

Este cliente está acessando o método buscarContato que contém os mesmos comandos da Figura 3.13 (Cliente.Inserção.java), porém ele seleciona o método buscarContato por meio do comando "*call.setOperationName()*", passa como parâmetro apenas o nome e imprime na tela o retorno da variável "ret", que está definida como um *array* do tipo *Object*, conforme mostrado na Figura 3.15.

Ao ser executado, ele busca o serviço buscarContato na interface WSDL e apresenta o seguinte resultado.

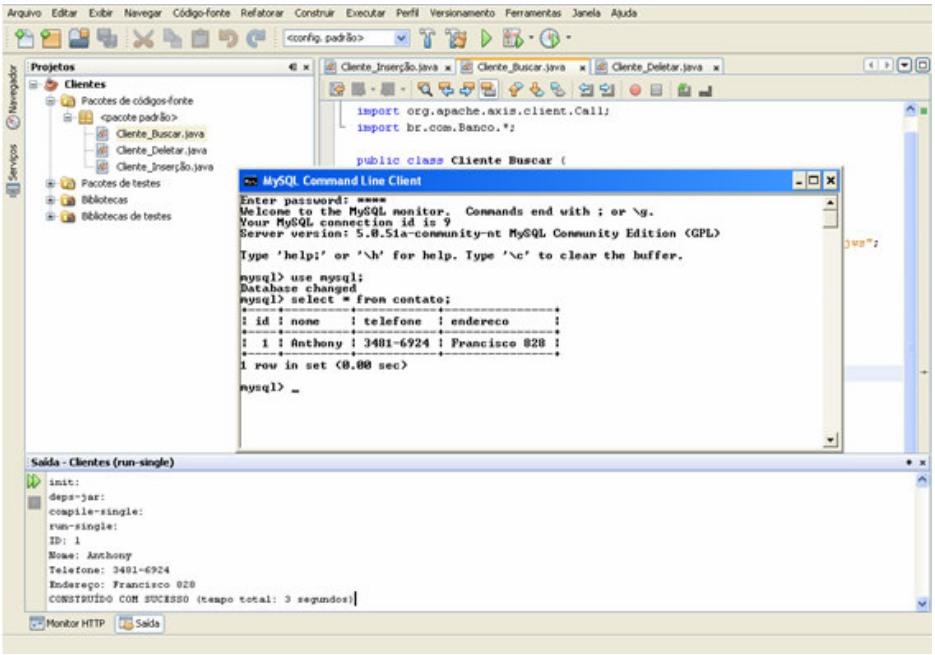

Figura 3.16 – Resultado da Busca

Pode-se observar que quando este serviço é acionado, ele retorna uma mensagem transformada no retorno do método invocado, que são todos os dados do nome do cliente que lhe foi passado por parâmetro, conforme ilustrado na Figura 3.16.

A seguir é mostrada a elaboração do serviço de remoção do Serviço Web.

```
public class Cliente_Deletar {
  public static void main(String[] args) throws Exception {
     // Endereço, local onde se encontra o Web Service
        String local = "http://localhost:8080/axis/Servico.jws";
     // Criando e configurando o serviço
        Call call = (Call) new Service().createCall();
     // Configurando o endereço.
        call.setTargetEndpointAddress(local);
     // Marcando o método a ser chamado.
        call.setOperationName("deletarContato");
          //Parâmetros da Função
        Object[] param = new Object[]{"Anthony"};
     //Retorno da Função
        Boolean ret = (Boolean)call.invoke(param);
     //Imprimindo o Resultado
        if(ret)
              System.out.println("Deletado");
        else
              System.out.println("Não Deletado");
  }
}
```

Figura 3.17 – Cliente_Deletar.java

Já neste Cliente, seleciona o método "deletarContato" por meio do método *"call.setOperationName()"*, passando como parâmetro um nome na variável "param" que é do tipo *array* de *Object*. Como retorno cria-se uma variável chamada "ret" do tipo boolean, fazendo-se uma condição *if* desta variável, caso ela retorne *true*, o registro é deletado, caso contrário, não. Ao ser executado este cliente busca o serviço deletarContato na interface WSDL e exibe o seguinte resultado.

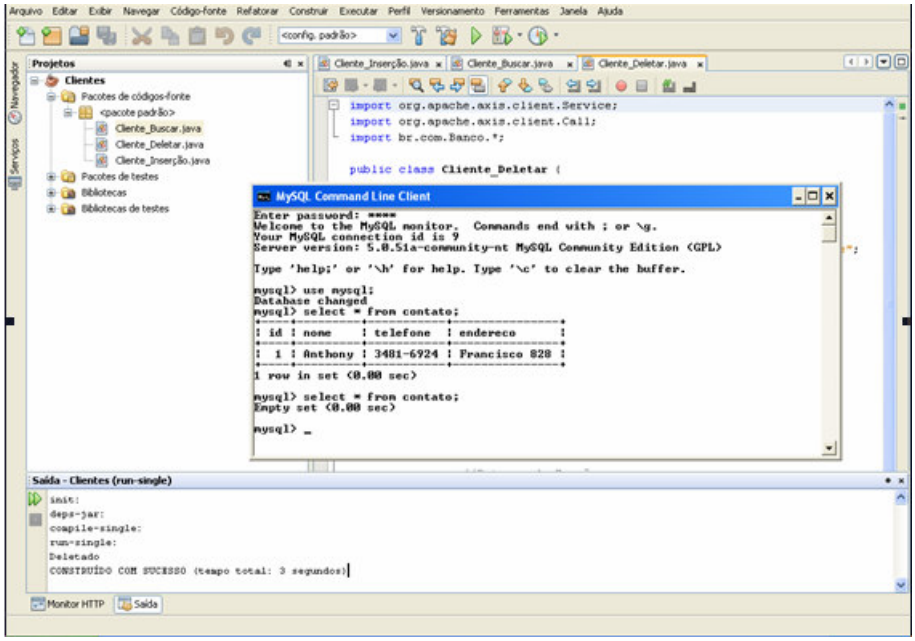

Figura 3.18 – Resultado da Remoção

Quando o serviço deletarContato é acionado no WSDL por este cliente, ele retorna uma mensagem que é transformada no retorno do método invocado, caso está mensagem seja *true*, ele informa que o nome que lhe foi passado por parâmetro foi deletado, caso seja *false,* ele informa que o contato não foi deletado, conforme ilustrado na Figura 3.18.

Apesar de conter três clientes e cada um acessar um serviço diferente, pode-se fazer um único cliente que acesse todos estes serviços oferecidos, ou melhor, uma aplicação que referencie o endereço deste Serviço Web (o endereço pode ser conseguido na Internet por meio da UDDI), lendo o seu descritor e utilizando dos seus serviços disponíveis, porem este foi feito separadamente para uma melhor compreensão de como um cliente acessa o serviço.

O Axis possui uma outra ferramenta que gera o código para o cliente acessar um Serviço Web, é o WSDL2Java. Ele cria um *ServiceLocator* e gera o *stub* para realização da chamada remota no servidor, mas para isto é necessário o *link* para o WSDL. Por exemplo, na ferramenta Eclipse pode-se usar o *plugin* WSDL2Java para importar para dentro de um projeto o descritor WSDL, que gera as classes num pacote padrão deste WSDL. A partir disto, pode-se usar o Serviço Web como classes que encapsulam as chamadas aos mesmos. Desta maneira é possível utilizar o auto-completar código do Eclipse, afinal o Serviço Web tornou-se classes dentro do projeto.

CONCLUSÕES

Pode-se observar que os Serviços Web possuem participações em projetos na qual necessitem proporcionar serviços a uma infinidade de aplicações, sendo elas aplicações Web ou locais.

A partir de então, observa-se por meio dos resultados obtidos, que os Serviços Web auxiliam na integração de sistemas e na comunicação entre aplicações diferentes, pelo fato de todas as aplicações envolvidas serem transmitidas por um protocolo padrão (SOAP) e por serem traduzidas para uma linguagem universal, o formato XML.

Mostrou-se que os Serviços Web pode prover agilidade para os processos e eficiência na comunicação entre aplicações distintas e que toda e qualquer comunicação entre aplicações passa a ser dinâmica e principalmente segura, pois não há intervenção humana.

Com a utilização do framework Axis foi possível mostrar de maneira simples e eficiente à construção de um Serviço Web, que a partir de uma plataforma Java, pode-se desenvolver um protótipo de uma aplicação *Web Service* capaz de entender o WSDL e o transformar em classes para utilização dos mesmos.

A partir do tutorial elaborado ao longo deste trabalho para o desenvolvimento de aplicações com Serviços Web, pode-se elaborar outras aplicações com funções diferentes, porém utilizando a mesma maneira que esta foi utilizada, pois o framework Axis pode ser considerado um framework ideal para a construção da mesma por apresentar funcionalidades que auxiliem e criem de forma automatizada os arquivos WSDL.

Foram pesquisas algumas maneiras de construção de Serviços Web, como por exemplo, a ferramenta JAX-WS e o REST do Net Beans, porém o framework Axis apresentou-se como uma alternativa mais fácil e objetiva de se criar Serviços Web, além de ser mais simples de se ensinar apresentando os mesmos resultados.

Serviços Web é a tecnologia ideal para comunicação entre sistemas, sendo muito utilizado em aplicações B2B (*Business to Business*), ou seja, utilizado por empresas, como por exemplo, a *Google* e empresas que fornecem leitura de cartões de créditos.

Portanto, neste contexto o framework Axis é considerado fundamental para o desenvolvimento de Serviços Web que é apresentado como a tecnologia ideal para fazer com que uma grande variedade de aplicativos, fornecedores e plataformas distintas se comunique por meio de protocolos e linguagens reutilizáveis, compreensíveis e padronizadas (SOAP e XML).

REFERÊNCIAS

ABINADER; Jorge; LINS; Rafael; **Web Service em Java**; BRASPORT; 2006.

ANDERSON, Robert. SEJUG – **Tutorial sobre Web Service Parte I**. Disponível em: http://www.sejug.org/docs/tutorialws.html. Acessado em 11 abr. 2008.

AOMESTRE; **Tutorial**; Disponível em http://www.aomestre.com.br/cyber/arquivo/ate2006/cyber21.htm . Acesso em 07 mar. 2008.

APACHE WEB SERVICES PROJECT. **Apache Axis2/Java**. Disponível via URL em http://ws.apache.org/axis2/ . Acessado em 20 abr. 2008 B.

APACHE WEB SERVICES PROJECT. **Apache XML-RPC**. Disponível via URL em http://ws.apache.org/xmlrpc/ . Acessado em 20 abr. 2008 A.

APOSTILANDO; **XML;** Disponível em http://www.apostilando.com . Acesso em 25 mar. 2008.

CAELUM, **Ensino e soluções em JAVA**. Disponível em www.caelum.com.br .Acessado em 08 nov. 2007.

CUNHA, Davi. Intelinet – **Web Service, SOAP e Aplicações Web**. Disponível em http://devedge-temp.mozilla.org/viewsource/2002/soap-overview/index_pt_br.html. Acessado em 05 fev. 2008

DÉCIO, Otávio C. **XML**: guia de consulta rápida. São Paulo: Novatec, 2000. 96p.

DEITEL; Harvey M.; DEITEL; Paul J.; **Java Como Programar 4ºEdição**, Bookman, 2003, 391p.

DESTRO, Daniel; **Tutorial do Apache Axis 1.3;** Disponível em http://www.guj.com.br/java.tutorial.artigo.180.1.guj . Acessado em 20 jul. 2008.

FARIA, Rogério A. **Treinamento avançado em XML**: Desvende os poderosos recursos desta linguagem. São Paulo: Digerati, 2005. 128p.

GOMES, Alexandre. Web Services: Simplicidade e integração com tecnologias abertas. **Java Magazine**, Rio de Janeiro: Neofício Editora Ltda., n. 1, p. 48-49, jul. 2002.

IWEB; **Serviços Web;** Disponível em http://www.iweb.com.br . Acesso em 15 dez. 2007.
OLIVEIRA; Robert; Anais dos **Serviços Web;** São Paulo, 2006.

JANDL JUNIOR, Peter. **Introdução ao Java**. São Paulo: Berkeley, 2002. 528p.

KABUTZ, Heinz. **Web Services**. Disponível em http://www.guj.com.br/java.tutorial.artigo.132.1.guj . Acessado em 10 dez. 2007

LOPES, **Web Services**, Disponível via URL em: http://www.fe.up.pt/~jlopes/jlopes@fe.up.pt . Acessado em 15 dez. 2007.

MARILIA; **Tutorial Web Service com Axis2 V1.4;** Disponível em http://datinf.cefetrn.br/doku.php?id=pessoal:corpodocente:marilia:disciplinas:2008.1:psd:web service. Acessado em 15 out. 2008.

MAHMOUD, Ousay H. **Service-Oriented Architecture (SOA) and Web Services**: The Road to Enterprise Application Integration (EAI). Disponível via URL em: http://java.sun.com/developer/technicalArticles/WebServices/soa/ . Acesso em 27 nov. 2007.

MCGOVER, J.; TYAGI, S.; STEVENS, M.; MATHEUS, S.; "**Java Web Services Architecture**"; Morgan Kaufmann; San Francisco – CA –USA, 2003;

MENDES, Fernando Vasconcelos. Web Services: dos conceitos à implementação. **Clube Delphi**, Rio de Janeiro: Neofício Editora Ltda., n. 27, p. 25-33, mai. 2002.

MIGUEL; **XML;** Disponível em http://www.gta.ufrj.br/grad/00_1/miguel . Acessado em 25 mar. 2008.

NETBEANS, **Tutorial Web Service**. Disponível em http://www.netbeans.org/kb/60/websvc/jax-ws.html . Acessado em 20 maio 2008.

OLIVEIRA, Eric. Linha de código – **Web Services**. Disponivel em http://linhadecodigo.com/Artigo.aspx?id=378. Acessado em 11 abr. 2008.

ORT, Ed. **Service-Oriented Architecture and Web Services**: Concepts, technologies, and tools. Disponível via URL em: http://java.sun.com/developer/technicalArticles/WebServices/soa2/ . Acessado em 27 nov. 2007.

RECKZIEGEL, Mauricio. **Descrevendo um Web Service**. Disponível em http://imasters.uol.com.br/artigo/4422/webservices/descrevendo_um_web_service_-_wsdl. Acessado em 05 mar. 2008.

RODRIGUES; Laércio; **Serviços Web**; Disponível em http://www.linhadecodigo.com/Artigo.aspx?id=778 . Acesso em 17 mar. 2008.

SAMPAIO, Cleuton. **SOA e Web Services em Java**. Rio de Janeiro: Brasport, 2006. 168p.

SUN MICROSYSTEMS. **The Java Web Services tutorial**. Disponível em: http://java.sun.com/webservices/docs/2.0/tutorial/doc/index.html . Acessado em 27 nov. 2007.

TECHNOMETRIA. **UDDI**. Disponível em http://photos.windley.com/gallery/view_photo.php?set_albumName=Misc&id=understanding_uddi . Acesso em 10 mar. 2008.

UDDI. **UDDI**. Disponível via URL em http://www.uddi.org/ . Acesso em 13 dez. 2007.

UNICAMP; **Serviços Web**; Disponível em http://libdigi.unicamp.br/document/?code=vtls000332721 . Acesso em 10 mar.2008.

VARCHAVSKY, Márcio; **Segurança para Web Service com WSS4J**; Revista Mundo Java; Edição 28; Ano V; 35p; 2008.

VELOSO, Renê R. **Java e XML**: Processamento de documentos XML com Java. Guia de consulta rápida. 2. ed. São Paulo: Novatec, 2007. 109p.

W3C. **WSDL**. Disponível via URL em http://www.w3.org/TR/wsdl20-primer/ . Acesso em 01 jan. 2008.

W3C. **Web Services**. Disponível via URL em: http://www.w3.org/2002/ws/ . Acesso em 04 dez. 2007.

APÊNDICE

Apêndice A - Classe Contato.java.

```
package br.com.Banco;
                    //Inicializando as variáveis do Banco de Dados

public class Contato {
private Integer id;
private String nome;
private String telefone;
private String endereco;

                        //Método construtor

    public Contato(Integer id, String nome, String telefone, String endereco)
{
        this.id = id;
        this.nome = nome;
        this.telefone = telefone;
        this.endereco = endereco;
    }
                         // Inicializando os Gets e Sets
    public Contato() {
    }
    public String getEndereco() {
        return endereco;
    }
    public Integer getId() {
        return id;
    }
    public String getTelefone() {
        return telefone;
    }
    public String getNome() {
        return nome;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public void setEndereco(String endereco) {
        this.endereco = endereco;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }
}
```

Apêndice B – Conexão.java

```
package br.com.FabricaDeConexao;

 import java.sql.*;
 public class Conexao {

      // Método que estabelece a conexão com o banco de dados

     public static Connection getConnection() throws SQLException{
         try{
             Class.forName("com.mysql.jdbc.Driver");
             return        DriverManager.getConnection("jdbc:mysql://localhost/
 mysql", "root", "root");
         }catch(ClassNotFoundException e){
             System.out.println("NÃ£o  foi  possivel  encontrar  o  driver  de
 banco: " + e.getMessage());
             throw new SQLException(e.getMessage());
         }
     }
  }
```

Apêndice C – ContatoDAO.java

```
 package br.com.Persistencia;

 import br.com.Banco.Contato;
 import java.sql.Connection;
 import java.sql.SQLException;
 import br.com.FabricaDeConexao.Conexao;
 import java.sql.PreparedStatement;
 import java.sql.ResultSet;
 import java.util.ArrayList;
 import java.util.List;
 import javax.swing.JOptionPane;

                     //Classe de Persistência

 public class ContatoDAO {
     private Connection connection;
     public ContatoDAO()throws SQLException{

             //Estabelece a conexÃ£o

         this.connection = Conexao.getConnection();
         System.out.println("CONECTADO");
     }

                 //Método que insere no Banco

     public void inserirContato(Contato contato) throws SQLException{
         PreparedStatement pre = this.connection.prepareStatement("insert into
 contato(id, nome, telefone, endereco) values(?, ?, ?, ?)");
         pre.setInt(1, contato.getId());
```

```
        pre.setString(2, contato.getNome());
        pre.setString(3, contato.getTelefone());
        pre.setString(4, contato.getEndereco());
        pre.execute();
        pre.close();
        connection.close();
        System.out.println("GRAVADO");
    }
          // Cria-se um Array do tipo Contato para Listar todos contatos da
busca

    public List<Contato> listarTodosContato() throws SQLException{
        List<Contato> contatos = new ArrayList<Contato>();
        PreparedStatement pre = this.connection.prepareStatement("select * from
contato");
        ResultSet rs = pre.executeQuery();
        while(rs.next()){
            contatos.add(new   Contato(rs.getInt("id"),   rs.getString("nome"),
rs.getString("telefone"), rs.getString("endereco")));
       }
        rs.close();
        pre.close();
        connection.close();
        return contatos;
    }

    public Contato buscarNome(String nomeContato) throws SQLException{
        PreparedStatement pre = this.connection.prepareStatement("select * from
contato where nome = ?");
        pre.setString(1, nomeContato);
        Contato newContato = new Contato();
        ResultSet rSet = pre.executeQuery();
        if(rSet.next()){
            newContato.setNome(rSet.getString("nome"));
            newContato.setEndereco(rSet.getString("endereco"));
            newContato.setTelefone(rSet.getString("telefone"));
            return newContato;
        }
        else
            return null;//se naun achar vai retornar null, sendo assim podemos
tratar isso no Servlet....ou atÃ© mesmo no jsp
    }

             // Método que deleta do Banco

    public void DeletarContato (Contato contato)throws SQLException {
        PreparedStatement  pre  =  this.connection.prepareStatement("delete  from
contato where nome = ?");
        pre.setString(1, contato.getNome());
        pre.execute();
        pre.close();
        connection.close();
        System.out.println("DELETADO");
    }
}
```

Apêndice D – Serviço.java

```
import br.com.Banco.Contato;
import br.com.Persistencia.ContatoDAO;
import java.io.*;
import java.net.*;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class Servico {
    public Object[] criarContato(int id, String nome, String telefone, String endereco) {
            Contato contato = new Contato(id, nome, telefone, endereco);
            try {
                    ContatoDAO contatoDao = new ContatoDAO();
                    contatoDao.inserirContato(contato);
            } catch (SQLException e) {
                    return null;
            }
            List<Object> contatoList = new ArrayList<Object>();
            contatoList.add(id);
            contatoList.add(nome);
            contatoList.add(telefone);
            contatoList.add(endereco);
            return contatoList.toArray();
    }

    public Object[] buscarContato(String nome) {
            try
            {
                    ContatoDAO contatoDAO = new ContatoDAO();
                    Contato contato = contatoDAO.buscarNome(nome);
                    if (contato != null)
                    {
                            List<Object> contatoList = new ArrayList<Object>();
                            contatoList.add(contato.getId());
                            contatoList.add(contato.getNome());
                            contatoList.add(contato.getTelefone());
                            contatoList.add(contato.getEndereco());
                            return contatoList.toArray();
                    } else
                            return null;
            } catch (SQLException e) {
                    return null;
            }
    }

    public boolean deletarContato ( String nome ) {
           try {
```

```
                    ContatoDAO contatoDao = new ContatoDAO();
                    Contato contato = contatoDao.buscarNome(nome);
                    if (contato!= null) {
                       contatoDao.DeletarContato(contato);
                       return true;
                    }
                    else
                            return false;
                    } catch (SQLException e) {
            }
            return true;
        }
}
```

Apêndice E – WSDL- Servidor

```
    <?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions        targetNamespace="http://localhost:8080/axis/Servico.jws"
    xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://localhost:8080/axis/Servico.jws"
    xmlns:intf="http://localhost:8080/axis/Servico.jws"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<wsdl:types>
<schema                  targetNamespace="http://localhost:8080/axis/Servico.jws"
    xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
<complexType name="ArrayOf_xsd_anyType">
<complexContent>
<restriction base="soapenc:Array">
  <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:anyType[]" />
    </restriction>
    </complexContent>
    </complexType>
    </schema>
    </wsdl:types>
<wsdl:message name="deletarContatoResponse">
  <wsdl:part name="deletarContatoReturn" type="xsd:boolean" />
    </wsdl:message>
<wsdl:message name="criarContatoRequest">
  <wsdl:part name="id" type="xsd:int" />
  <wsdl:part name="nome" type="xsd:string" />
  <wsdl:part name="telefone" type="xsd:string" />
  <wsdl:part name="endereco" type="xsd:string" />
    </wsdl:message>
<wsdl:message name="deletarContatoRequest">
  <wsdl:part name="nome" type="xsd:string" />
    </wsdl:message>
<wsdl:message name="buscarContatoResponse">
  <wsdl:part name="buscarContatoReturn" type="impl:ArrayOf_xsd_anyType" />
```

```
  </wsdl:message>
wsdl:message name="buscarContatoRequest">
 <wsdl:part name="nome" type="xsd:string" />
  </wsdl:message>
wsdl:message name="criarContatoResponse">
 <wsdl:part name="criarContatoReturn" type="impl:ArrayOf_xsd_anyType" />
</wsdl:message>
sdl:portType name="Servico">
sdl:operation name="criarContato" parameterOrder="id nome telephone endereco">
<wsdl:input message="impl:criarContatoRequest" name="criarContatoRequest" />
<wsdl:output message="impl:criarContatoResponse" name="criarContatoResponse" />
</wsdl:operation>
sdl:operation name="buscarContato" parameterOrder="nome">
<wsdl:input message="impl:buscarContatoRequest" name="buscarContatoRequest" />
<wsdl:output message="impl:buscarContatoResponse" name="buscarContatoResponse" />
</wsdl:operation>
sdl:operation name="deletarContato" parameterOrder="nome">
<wsdl:input message="impl:deletarContatoRequest" name="deletarContatoRequest" />
<wsdl:output message="impl:deletarContatoResponse" name="deletarContatoResponse" />
</wsdl:operation>
</wsdl:portType>
sdl:binding name="ServicoSoapBinding" type="impl:Servico">
 <wsdlsoap:binding  style="rpc"  transport="http://schemas.xmlsoap.org/soap/http"  />
<wsdl:operation name="criarContato">
<wsdlsoap:operation soapAction="" />
sdl:input name="criarContatoRequest">
     <wsdlsoap:body      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded" />
</wsdl:input>
wsdl:output name="criarContatoResponse">
        <wsdlsoap:body   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/Servico.jws" use="encoded" />
</wsdl:output>
</wsdl:operation>
sdl:operation name="buscarContato">
<wsdlsoap:operation soapAction="" />
sdl:input name="buscarContatoRequest">
       <wsdlsoap:body    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded" />
</wsdl:input>
sdl:output name="buscarContatoResponse">
       <wsdlsoap:body    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/Servico.jws" use="encoded" />
</wsdl:output>
</wsdl:operation>
sdl:operation name="deletarContato">
<wsdlsoap:operation soapAction="" />
wsdl:input name="deletarContatoRequest">
       <wsdlsoap:body    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://DefaultNamespace" use="encoded" />
</wsdl:input>
wsdl:output name="deletarContatoResponse">
      <wsdlsoap:body     encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/Servico.jws" use="encoded" />
```

```
/wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
sdl:service name="ServicoService">
sdl:port binding="impl:ServicoSoapBinding" name="Servico">
 <wsdlsoap:address location="http://localhost:8080/axis/Servico.jws" />
   </wsdl:port>
   </wsdl:service>
    </wsdl:definitions>
```

Apêndice F – Cliente_Inserção.java

```
    import org.apache.axis.client.Service;
    import org.apache.axis.client.Call;
    import br.com.Banco.*;

    public class Cliente_Inserção {
      public static void main(String[] args) throws Exception {
            // Endereço, local onde encontra-se o Web Service
              String local = "http://localhost:8080/axis/Servico.jws";
            // Criando e configurando o serviço
              Call call = (Call) new Service().createCall();
            // Configurando o endereço.
              call.setTargetEndpointAddress(local);
            // Marcando o método a ser chamado.
              call.setOperationName("criarContato");
            // Parâmetros da função criarContato.
              Object[]   param   =   new   Object[]{1,   "Anthony",   "3481-6924",
    "Francisco 828"};
            // Retorno da Função
              Object[] ret = (Object[])call.invoke(param);
            // Imprime o resultado
              System.out.println(" Inserido com Sucesso " );
      }
    }
```

Apêndice G – WSDL- Cliente_Inserção

```
  <?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
<criarContatoResponse
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<criarContatoReturn   soapenc:arrayType="xsd:anyType[4]"   xsi:type="soapenc:Array"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
  <criarContatoReturn xsi:type="soapenc:int">1</criarContatoReturn>
  <criarContatoReturn xsi:type="soapenc:string">Anthony</criarContatoReturn>
  <criarContatoReturn xsi:type="soapenc:string">3481-6924</criarContatoReturn>
    <criarContatoReturn
```

```
    xsi:type="soapenc:string">Av.Republica</criarContatoReturn>
   </criarContatoReturn>
  </criarContatoResponse>
  </soapenv:Body>
  </soapenv:Envelope>
```

Apêndice H – Cliente_Buscar.java

```
    import org.apache.axis.client.Service;
    import org.apache.axis.client.Call;
    import br.com.Banco.*;

    public class Cliente_Buscar {
      public static void main(String[] args) throws Exception {
            // Endereço, local onde encontra-se o Web Service
              String local = "http://localhost:8080/axis/Servico.jws";
            // Criando e configurando o serviço
              Call call = (Call) new Service().createCall();
            // Configurando o endereço.
              call.setTargetEndpointAddress(local);
            // Marcando o método a ser chamado.
              call.setOperationName("buscarContato");
            //Parâmetros da Função
              Object[] param = new Object[]{"Anthony"};
            //Retorno da Função
              Object[] ret = (Object[])call.invoke(param);
            //Imprimindo o Resultado
              System.out.println("ID:  "  +  ret[0]  +  "\nNome:  "  +  ret[1]  +
    "\nTelefone: " + ret[2]+ "\nEndereço: " + ret[3]);
        }
       }
```

Apêndice I – WSDL – Cliente_Buscar

```
  <?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
<buscarContatoResponse
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<buscarContatoReturn  soapenc:arrayType="xsd:anyType[4]"  xsi:type="soapenc:Array"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
  <buscarContatoReturn xsi:type="soapenc:int">1</buscarContatoReturn>
  <buscarContatoReturn xsi:type="soapenc:string">Anthony</buscarContatoReturn>
  <buscarContatoReturn xsi:type="soapenc:string">34816924</buscarContatoReturn>
                                                          <buscarContatoReturn
    xsi:type="soapenc:string">Av.Republica</buscarContatoReturn>
    </buscarContatoReturn>
    </buscarContatoResponse>
    </soapenv:Body>
    </soapenv:Envelope>
```

Apêndice J – Cliente_Deletar.java

```
import org.apache.axis.client.Service;
import org.apache.axis.client.Call;
import br.com.Banco.*;

public class Cliente_Deletar {
   public static void main(String[] args) throws Exception {
         // Endereço, local onde encontra-se o Web Service
           String local = "http://localhost:8080/axis/Servico.jws";
         // Criando e configurando o serviÃ§o
           Call call = (Call) new Service().createCall();
         // Configurando o endereço.
           call.setTargetEndpointAddress(local);
         // Marcando o método a ser chamado.
           call.setOperationName("deletarContato");
         //Parâmetros da FunÃ§Ã£o
           Object[] param = new Object[]{"Anthony"};
         //Retorno da Função
           Boolean ret = (Boolean)call.invoke(param);
         //Imprimindo o Resultado
           if(ret)
                 System.out.println("Deletado");
           else
                 System.out.println("NÃ£o Deletado");
     }
  }
```

Apêndice K – WSDL – Cliente_Deletar

```
  <?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope       xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
<deletarContatoResponse
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <deletarContatoReturn xsi:type="xsd:boolean">true</deletarContatoReturn>
    </deletarContatoResponse>
    </soapenv:Body>
    </soapenv:Envelope>
```